Federated E-infrastructure Dedicated to European Researchers Innovating in Computing network Architectures

Co-funded by the European Commission within the Seventh Framework Programme. Grant Agreement No: RI-213107

Deliverable DJRA1.2
Solutions and protocols proposal for the network control, management and monitoring in a virtualized network context

Version DJRA1.2v2.1

Dissemination Level:	Public
Contractual date of deliverable:	August 31st, 2009
Actual submission date:	October 30th, 2009
Editor's Names:	Cristina Cervelló-Pastor (UPC) Robert Machado (UPC) Álvaro Monje (UPC) Ásgeir Óskarsson (UPC)

Partners who have contributed to this document:

Partner	Name	E-mail address
GARR	Ugo Monaco	ugo.monaco@garr.it
DFN	Peter Kauffman	kaufmann@dfn.de
	Monika Roesler	roesler@dfn.de
Uni Erlangen	Susanne Naegele-Jackson	Susanne.Naegele-Jackson@rrze.uni-erlangen.de
GRNet	Constantinos Vassilakis	cvassilakis@grnet.gr
PSNC	Łukasz Dolata	ldolata@man.poznan.pl
i2CAT	Sergi Figuerola	sergi.figuerola@i2cat.net
	Josep Pons	josep.pons@i2cat.net
KTH	Markus Hidell	mahidell@kth.se
	Peter Sjödin	psj@kth.se
	Pehr Söderman	pehrs@kth.se
ICCS	Dimitris Kalogeras	D.Kalogeras@noc.ntua.gr
UPC	Cristina Cervelló-Pastor	cristina@entel.upc.edu
	Roberto Machado	robert.machado@entel.upc.edu

UPC	**Álvaro Monje** **Ásgeir Óskarsson** **Sebastià Sallent**	*alvaro.monje@entel.upc.edu* *asgeir.oskarsson@entel.upc.edu* *sallent@entel.upc.edu*
Juniper	**Jean-Mark Uze**	*juze@juniper.net*

Abstract

This deliverable presents several research proposals for the FEDERICA network, in different subjects, such as monitoring, routing, signalling, resource discovery, and isolation. For each topic one or more possible solutions are elaborated, explaining the background, functioning and the implications of the proposed solutions.

Document Revision History

Version	Date	Description of change	Author
0.9	July 17th 2009	Changes made to all chapters	UPC
1.2	July 29th 2009	Inter-domain, chapters updated	UPC
1.4	August 26th 2009	Global revision Resource Description updated Resource Allocation added Monitoring added Implementation of OpenFlow added User Portal updated Isolation updated	UPC, ICCS i2CAT&UPC GRNET GARR/Juniper DFN / Uni Enlargen PSNC KTH
1.5	August 31th 2009	Implementation of OpenFlow updated. Global revision	DFN / Uni Enlargen UPC ICCS
1.8	September 30th 2009	Global revisión Abstract, conclusions added	UPC

Table of Contents

List of Figures

List of Tables


FEDERICA
WWW.FP7-FEDERICA.EU

1 Introduction

1.1 Purpose and Scope

The previous deliverable (DJRA1.1, [2]) was focused on the design of the building blocks needed to define a virtual architecture. In that deliverable we evaluated several tools and frameworks that could be used for tool bench development or to create virtual slices. Four different tools were identified as most adequate for FEDERICA: Manticore with IaaS, BLUEnet Tool, DRAGON, and PL-VINI.

This deliverable goes further on the research aspects within FEDERICA. First of all the architecture of the control plane for the FEDERICA infrastructure will be defined. Several possibilities could be implemented, using the basic FEDERICA infrastructure as a starting point. The focus on this document is the intra-domain aspects of the control plane and their properties. Also some inter-domain aspects are addressed.

The main objective of this deliverable is to lay great stress on creating and implementing the prototype/tool for the FEDERICA slice-oriented control system using the appropriate framework. This deliverable goes deeply into the definition of the containers between entities and their syntax, preparing this tool for the future implementation of any kind of algorithm related to the control plane, for both to apply UPB policies or to configure it by hand. We opt for an open solution despite the real time limitations that we could have (for instance, opening web services connexions or applying fast recovering mechanisms).

The application being developed is the central element in the control plane, and additional features must be added to this application. This control plane, from the functionality point of view, is composed by several procedures that provide a reliable application and that include some mechanisms or algorithms to be able to discover and assign resources to the user.

To achieve this, several topics must be researched in order to propose new protocols for the virtual infrastructure. The topics and necessary features covered in this document include resource discovery, resource allocation, signalling, routing, isolation and monitoring. All these topics must be researched in order to find a good solution for the FEDERICA network. Some of these algorithms have started to be analyzed and will be expanded in the next deliverable. Current standardization and existing solutions have been investigated in order to find a good solution for FEDERICA. Resource discovery is an important issue within the FEDERICA network, as manual resource discovery is no option, due to scalability requirement. Furthermore, no standardization exists, so knowledge must be obtained from related work. Ideally, the proposed solutions for these topics should not only be adequate specifically for this infrastructure, but could also be applied to other virtualized networks.

This deliverable lays the foundations for the protocols proposals for network control, management and monitoring in a virtualized network context. Eventually, these

protocol proposals should be implemented in prototypes. During the prototyping process these proposals will be shaped and sometimes adapted, but a good basis of all protocols is already defined. This prototyping is not included in the scope of this deliverable, but will be for deliverable DJRA1.3.

1.2 Document overview

This document is structured as follows. First an overview of the FEDERICA architecture is given. Both the network architecture and the control-plane specific architecture will be discussed in Chapter 0. After the architecture is presented, all relevant concepts are introduced bottom-up. Chapter 3 is about the resource descriptions. In this chapter an overview of all FEDERICA resources will be given. From chapter 4 on the document emphasizes more on theoretical research. Chapter 4 introduces resource discovery and allocation mechanisms which might be implemented within FEDERICA. Chapter 4 also goes in the signalling implications within FEDERICA. This chapter covers signalling messages from several network elements. Chapter 5 introduces mechanisms which might be implemented for routing in FEDERICA and introduces the OpenFlow implementation as a new network architectural platform that intends to simplify network control and management. Chapter 6 explores the monitoring aspects of the FEDERICA research. Chapter 7 presents the isolation topic. Chapter 8 deals with the inter-domain implications of the different mechanisms presented in this document. Chapter 9 is about the User Portal, a tool which is being developed to offer the different functionalities of the FEDERICA control and management plane to the users. Finally, the conclusions are presented in chapter 10.

1.3 List of Abbreviations

AAI – Authentication and Authorization Infrastructure

API – Application Programming Interface

BWCTL - Bandwidth test Controller

CLI – Command Line Interface

DRAGON - Dynamic Resource Allocation via GMPLS Optical Networks

DTD - Document Type Definition

EPR – EndPoint Reference

ESFS – Ethernet Switch Factory Service

FEDERICA – Federated E-infrastructure Dedicated to European Researchers Innovating in Computing network Architectures

FSMS – FEDERICA Slice Monitoring System

FUP – FEDERICA User Portal

GQAP - Generalized Quadratic Assignment Problem

GUI – Graphical User Interface

HADES - HADES Active Delay Evaluation System

IaaS – Infrastructure as a Service

IETF – International Engineering Task Force

IS-IS – Intermediate System to Intermediate System

JMX – Java Management Extensions

LDAP – Lightweight Directory Access Protocol

LSP – Label Switched Path

LSR – Label Switching Router

MOPSO – Multi Objective Particle Swarm Optimization

MRGAP – Multi-Resource Generalized Assignment Problem

MR-GQAP – Multi-Resource Generalized Quadratic Assignment Problem

MTU – Maximum Transmission Unit

NDL – Network Description Language

NFS – Network File System

NIC – Network Interface Card

NOC – Network Operating Center

NSIS - Next Steps In Signalling

OSPF – Open Shortest Path First

PCE – Path Computation Element

PL-VINI – PlanetLab Virtual Network Infrastructure

PoP – Point of Presence

PSO - Particle Swarm Optimization

QoS – Quality of Service

RCP - Rich Client Platform

RFC – Request For Comments

RSVP – Resource reSerVation Protocol

SDK – Software Development Kit

SIP – Session Initiation Protocol

SNMP – Simple Network Management Protocol

SOAP – Simple Object Access Protocol

SSH – Secure Shell

TRILL – Transparent Interconnection of Lots of Links

TTL – Time To Live

UPB – User Policy Board

URI – Uniform Resource Identifier

VI – Virtual Infrastructure

VLAN – Virtual Local Area Network

VN – Virtual Network

VNE – Virtual Network Embedding

VPN – Virtual Private Network

WAN – Wide Area Network

WSDL – Web Service Description Language

WSRF – Web Service Resource Framework

XML – Extensible Mark-up Language

XORP – Extensible Open Router Platform

XSD – XML Schema Definition

2 FEDERICA Architecture

2.1 Control Plane Overview

In previous stages the FEDERICA infrastructure has been developed. The figure below gives an overview of the overall infrastructure of the FEDERICA network. Currently the network consists of thirteen Points of Presence (PoPs). Four of these are core PoPs which are connected in a mesh network: GARR, DFN, PSNC, and CESNET. These connections are depicted with red lines in the Fig. 2-1.

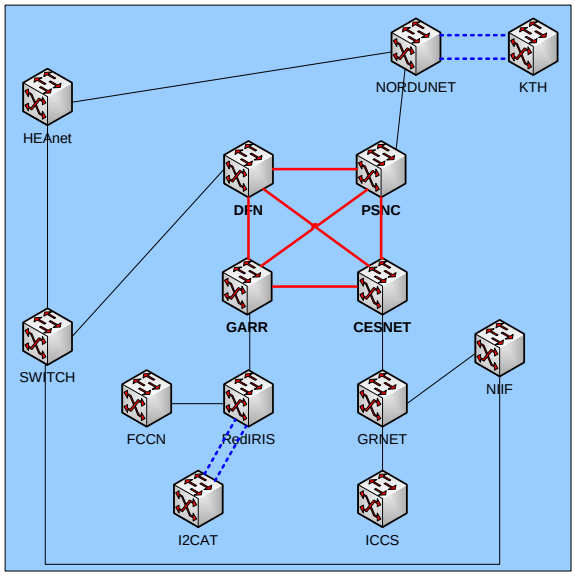

Fig. 2-1. FEDERICA Topology [1]

The figure above (Fig. 2-1) depicts the data plane connections between the different PoPs. Each switch, router and/or virtual machine inside the PoPs is assigned an Engine in the control plane, which controls the virtualization of the devices. These Engines are placed locally at the devices and are responsible for the communication between the devices and the Network Service. The Network Service communicates requests for resources and specifies the ports which the slices should use. The Resources table has all the resources stored and communicates with the Network Service on the specific ports and resources being used for slices.

At the moment, this is the basic functionality of the control plane for the arrangement of virtual slices. Further details on, among others, signalling have not yet been defined. These functionalities will become clearer as the project develops. The figure below

(**Error! Reference source not found.**) depicts the overall functionality of the Network Service, which is located in the control plane.

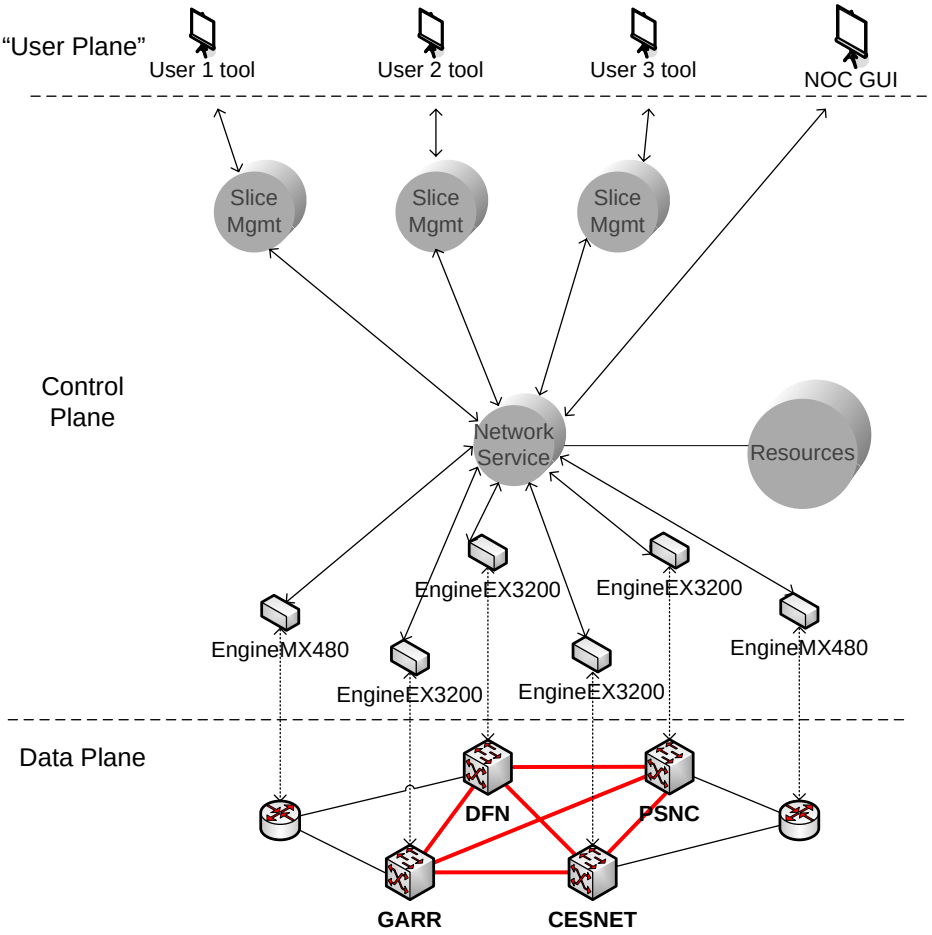

Fig. 2-2. Control Plane Overview

In a multi-domain scenario the control plane will try to maintain the coherence in the devices located along the different domains. In order to ensure the end-to-end implementation of the virtual infrastructure, the control plane will have to be aware of the available virtual resources along all the way in the different domains.

There are two different possibilities for the control plane architecture. On one hand, a Network Service per domain could be implemented. The communication among the Network Services governing each domain should be defined to achieve this multi-domain coherence. Nevertheless, this is only one of the two possibilities to implement.

On the other hand, there is a centralized model where the resources of the different domains are managed in a centralized way. In this situation, a unique Network

Service would manage the resources in the different domains. A centralized Network Service is showed in the Fig. 2-3. In this case, a slice management mechanism that maintains the configurations for the different slices is considered and each slice management service communicates with the Network Service.

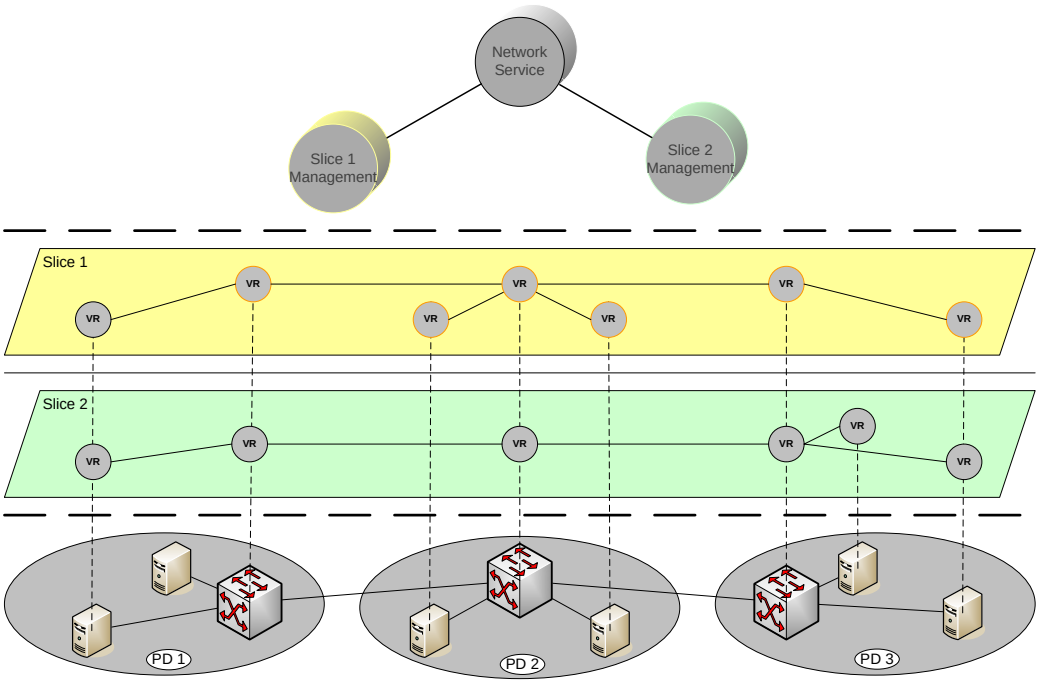

Fig. 2-3. Multi-domain FEDERICA Overview

2.2 Control Plane Application

In the current application version the logic of the tool bench is stored in the GUI. Resources must be added and modified manually, and no automated processes are supported. The first proposition is to develop a Network Service that will take over some of the logic and be able to automate the tasks of the GUI. To manage the slices, slice management functionality must be added. Manticore prototype version has a similar functionality for L3, with virtualized IP networks, but this functionality is not yet available for L2.

The resources used to build the Federica Management Network are not based on Federica Data Plane. Therefore, current version of the tool bench redirects the traffic over the Internet, and does not use the FEDERICA network to connect to the resources in the different physical domains. Another possibility would be to guide this traffic over the FEDERICA network, but at the moment this is not the case. The figure below (Fig. 2-4) shows the current implementation, including the proposed

Network Service and Slice Management. All components of the application will be described briefly in this section.

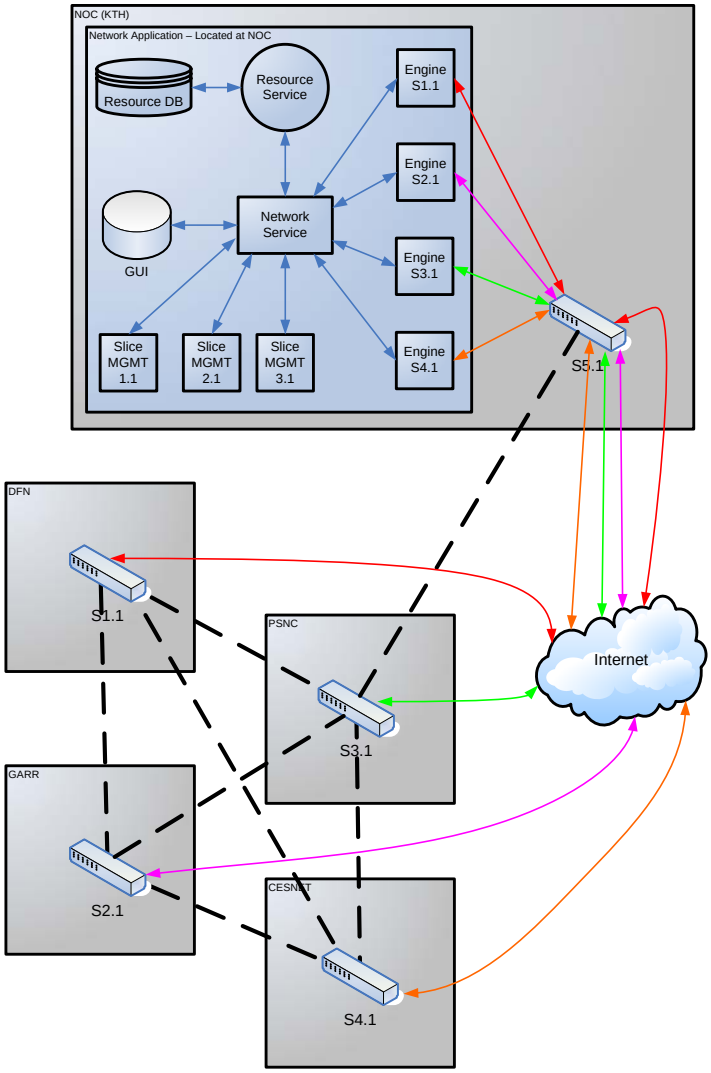

Fig. 2-4. Network Application Overview

- **GUI**: The Graphical User Interface currently implements all the logic for manual configuration of the settings for the physical and virtual resources. The GUI calls web services to communicate with the physical resources.
- **Network Service**: This is the new central service which coordinates the automatic configuration of the physical and virtual resources.

- **Resource Service**: This web service is called by the Network Service whenever it needs information from the resource database. Its functionality is purely based on the storage and communication of resources in the database.
- **Resource Database**: A PostgreSQL database which includes all the network resources and their configurations.
- **Slice Management**: The slice management communicates with the Network Service to obtain the configurations for a slice. For each slice a new slice management must be created.
- **Engine**: Network Service will be responsible for its creation. The engine communicates via Netconf sessions with the physical resources, and communicates the configurations to the Network Service.

Graphical User Interface

The graphical user interface (GUI) is a simple interface displaying the main actions and capabilities of the tool. This tool is based on the Eclipse Rich Client Platform (RCP), a set of libraries that offers the developer the possibility of managing components to be used to form a final rich client application (similar to Eclipse). The interface offers the view of the devices and the resources configured in each of them. The user is able to add new devices to the tool, thanks to the wizards that will guide him. In each one of these configured devices, the user will be able to see and configure the VLANs and the interfaces.

This GUI presents some functionality limitations that will have to be solved. For instance, the application is able to assign multiple interfaces to a VLAN at time, while the user interface only allows one at time. Improving this in the graphical interface would reduce the configuration time, because just one connection to the switch would be necessary, instead one connection for each action.

Eclipse RCP applications can be presented as stand-alone applications or as Eclipse Plug-in as well. In this case, due to the early stage in the development of the user interface, the application is presented as an Eclipse plug-in.

Network Service

The network service must be the core service of the application. This part of the tool will on one hand be responsible for the resource and device services, mapping these into different Slice objects. The Network Service will then be responsible for the communication with the devices engines. It will also be responsible for the management of the virtualized resources, and the communication with the Resources Service.

The user tools may access these slice managers in order to manage the user resources (for instance, create their own VLANs). The NOC GUI will access Network service in order to keep its inventory updated.

Resource Service

The main functionality of the resource service is to maintain communications between the network service and the resource database. Requests by the network service to store, transmit or adjust configurations in the resource database will be handled by the resource service. It is a web service that is able to communicate with the resource database in SQL.

Resource Database

The resource database is located in the same place as the resource service and the other services inside the network application. This is not a fixed setting, as it could also be stored at a different location. The resource database is based on PostgreSQL, which is an open source database, and communicates directly with the resource service in this language. The database consists of seven tables, which shall be explained briefly in this paragraph. The table that represents the switches, *Ethernet_switch_resources*, contains the main characteristics of the FEDERICA network switch. It has a unique *resourceKey* and unique name, along with some other characteristics, which can be seen in Table 2-1.

Table 2-1. Ethernet Switch Resources

Table: Ethernet_switch_resources		
Column	Type	Description
resourceKey	String	Primary Key
version	Integer	Version number
name	String	Switch name (unique)
model	String	Switch model
manufacturer	String	Switch manufacturer
destructionTime	Long	Switch lifetime

The *resourceKey* is the primary identifier for a physical switch. The version gives the version number of the switch. The switch name must be unique for every switch, as it is used by another table in the database. The model is the model of the switch, which in the case of FEDERICA is EX3200. The manufacturer for the FEDERICA switches is Juniper. At the moment no other switches but the Juniper EX3200 are supported by the FEDERICA control plane. The *destructionTime* is not used for the moment, as it does not apply to the core FEDERICA switches. If in the future other switches will be added to the network, this option can be used to adapt to specific requirements of the user.

The primary resource key is coupled to the *switch_configModules* and *switch_configParameters* tables, where it is a foreign key. The *switch_configModules* table contains one additional field, which carries the transport and the protocol used. At the moment, as the connections between the switch and engines use these protocols, this field will contain SSH as Reliable transport protocol and NetConf as protocol. This table is shown in Table 2-2.

Table 2-2. Switch_ConfigModules

Table: Switch_configModules		
Column	Type	Description
resourceKey	String	Foreign Key
configModules	String (1000)	Transport & Protocol

The *switch_configParameters* contains all the parameters needed to access the switch. Inside the field *configParameters* settings as IP address, username and password are stored. The table can be seen below, in Table 2-3.

Table 2-3. Switch_ConfigParameters

Table: Switch_configParameters		
Column	Type	Description
resourceKey	String	Foreign Key
Configparameters	String (1000)	Contains Parameters

The database also contains a table which describes all the characteristics of Ethernet resources. This table, *ethernet_resources*, is linked to the *ethernet_switch_resources* table by its Resource Key and the unique name of the switch table. At the moment, the *ethernet_resources* table consists of all the characteristics of a port, but other resources could also be included, such as tunnels, OSPF, etc. The table can be seen below, in Table 2-4.

Table 2-4. Ethernet Resources

Table: Ethernet_resources		
Column	Type	Description
resourceKey	String	Primary Key
version	Integer	Version number
owner	String	Username
connectedTo	String	Connection
ipAddressV4	String	IPv4 Address
ipAddressV6	String	IPv6 Address
netmaskV4	String	Netmask
isTrunk	Boolean	Trunk or Access
Type	String	Port Type
MTU	String	Max. Transmission Unit

VCATProperties	String	VCAT Properties
LCASProperties	String	LCAS Properties
Direction	String	Direction
crossConnectionDescription	String	Cross Connection
Status	String	Status
Reason	String	Reason
Description	String	Description
networkElement	String	Network Element
physicalbinding	String	Linked switch resource
destructionTime	Long	Resource lifetime

Apart from the connection with the *ethernet_switch_resources* table, it is connected to three different tables. These connections are based upon the primary resource key of the *ethernet_resources*, which is coupled to the foreign resource key in the *subtables vlans, crossconnected* and extension. Multiple VLANs can be connected to a single port, where the column vlan stores the VLAN ID. This table is shown in Table 2-5.

Table 2-5. Ethernet Resources VLANs

Table: Ethernetresource_vlans		
Column	Type	Description
resourceKey	String	Foreign Key
vlan	String (1000)	VLAN ID

A single port can have multiple connections, which are stored in the table *ethernetresource_crossconnected*. The field *crossConnected* contains all the connections a port has with other ports. The table is illustrated below in table 2.6.

Table 2-6. Ethernet Resource CrossConnected

Table: Ethernetresource_crossconnected		
Column	Type	Description
resourceKey	String	Foreign Key
crossConnected	String (1000)	Port Connections

The last table in the resource database is the port extension table, called *ethernetresource_extension*. This table allows for possible extensions of the parameters, so at the moment is not used in the resource database. Table 2-7 illustrates this table.

Table 2-7. Ethernet Resource Extension

Table: Ethernetresource_extension		
Column	Type	Description
resourceKey	String	Foreign Key
extension	String (1000)	Possible extensions

The relationships between the object-oriented resource database tables are shown below in Fig. 2-5. It becomes clear that one switch can contain multiple ports, parameters and modules. Furthermore, one port can contain multiple cross connections, VLANs or extensions.

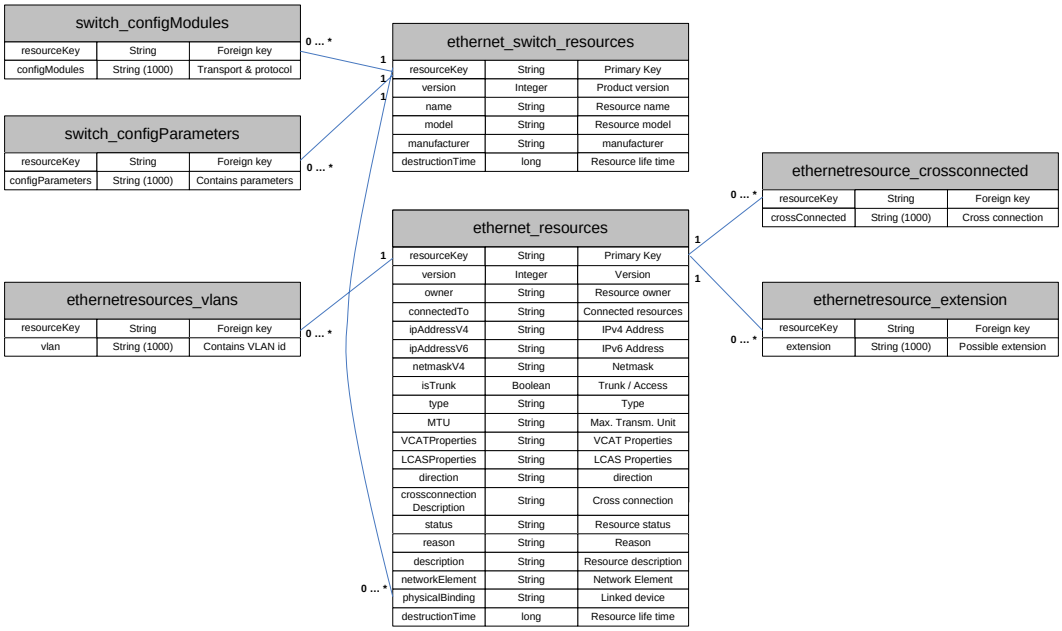

Fig. 2-5. Relationships between tables in resource database

3 Resource Descriptions

3.1 Overview of FEDERICA resources

Since the virtualization and the sharing of network resources are two of the main objectives of the FEDERICA deployment, the correct characterization and description of these resources is a key issue to be specified. This would ensure an easy implementation of the tool and the different protocols that will manage these resources.

In order to retrieve the resource description of the various devices, a set of protocols and languages are used. On one hand, information from switches and routers is obtained using NetConf. On the other hand the management of V-nodes is performed by VI-API. Both mechanisms (NetConf and VI-API) are offered by the switches/routers and V-nodes respectively. With these mechanisms, FEDERICA can reach an appropriate resource description of its infrastructure.

The resource description that has been followed is conditioned by the XML resource description that the network devices offer. This description is modelled in the tool in a very similar way, trying to the keep the devices description format. In that sense, the description of the resources for the communication between the modules of the tool is done in Web Service Description Language (WSDL), which is based on XML, also in a very similar way as the switch or routers showed information. Web services must often provide their users with the ability to access and manipulate state, i.e., data values that persist across, and evolve as a result of, Web service interactions. And while Web services successfully implement applications that manage state today, we need to define conventions for managing state so that applications discover, inspect, and interact with stateful resources in standard and interoperable ways. The WS-Resource Framework (WSRF) defines these conventions and does so within the context of established Web services standards (i.e. WSDL, WS-addressing etc). Globus Toolkit is a Framework that implements WSRF mechanisms for creating Web Service based distributed applications [3]. Because Manticore Web Services are also based on the WSRF standard, FEDERICA's resource description will also use these mechanisms.

Other possibilities such as Network Description Language (NDL) have also been studied. However, due to the compatibility inconsistency of NDL with Manticore prototype, WSRF standard and WSDL descriptions are used to obtain Layer 2 resource information and configuration with the objective to obtain a proper resource description.

3.2 NetConf and JUNOS XML API

3.2.1 Introduction

Netconf is an API based on XML (Extensible Markup Language). Client applications use this API to exchange information with the Netconf server running on the switch or router. The API provides a way to display, edit and commit configuration statements. The requests and responses sent/received using the API are equivalent as sets of configuration commands using the command line interface (CLI).

The JUNOS XML API represents the JUNOS configuration statements and operational mode commands in XML. The operations of the NETCONF API respond to the JUNOS XML API tags.

Working with Netconf and JUNOS XML APIs offers some advantages. One of them is that both are programmable interfaces, and they fully document all options for every supported JUNOS operational request and all elements in every configuration statement.

Other advantage is the simplicity to understand the content and structure of the XML-tagged data sets, due to the combination of the meaningful tag names and the structural rules in a DTD (Document type definition).

Other of the main advantages of Netconf is that it is able to get the status of the switch, and parse it to the program in an easier way than CLI would do. In the following lines there are some examples of configuration of the switch using Netconf.

3.2.2 Language description

To map commands to JUNOS XML Tag Elements the JUNOS API defines tag-element equivalents for many of those commands in CLI operational mode.
Many CLI commands have options that identify the object that the command affects or reports about, distinguishing the object from other objects of the same type. Moreover, many commands include options that have a fixed form which specify the amount of detail to include in the output. JUNOS XML API usually maps such an option to an empty tag whose name is the option name.

3.2.3 Resources request

The request message sent to the devices is the following:

```
<rpc>
   <get-config>
      <source>
```

```
            <candidate/>
        </source>
        <filter>
            <configuration>
                <interfaces></interfaces>
                <vlans></vlans>
            </configuration>
        </filter>
    </get-config>
</rpc>
```

With the specification of a filter as is shown below, we limit the information asked to the switch to the elements included within the filter. In this case, only interfaces and VLANs information will be retrieved.

```
        <filter>
            <configuration>
                <interfaces></interfaces>
                <vlans></vlans>
            </configuration>
        </filter>
```

3.2.4 Interfaces description

Once the switch sends the configuration information, we can obtain the resources description from the tag hierarchy as it is explained below.

The interfaces description is done as shown in the following code:

```
<interfaces>
    <interface>
      <name>ge-0/0/0</name>
        <unit>
          <name>0</name>
          <description></description>
          <family>
            <ethernet-switching>
              <port-mode>trunk</port-mode>
              <vlan-id>120</vlan-id>
            </ethernet-switching>
          </family>
        </unit>
    </interface>
</interfaces>
```

All the described interfaces are encapsulated by the <interfaces></interfaces> tags. Inside these tags, each interface is described, surrounded by the <interface></interface> tags. Inside these tags the interface properties are described by the following tags.

```
<name></name>
```

Inside each interface, units are described, specified among <unit></unit> tags.

The properties of each unit are described by the following tabs:

```
<name></name>
<description></description>
<family></family>
```

Inside family tabs, more family-specific properties can be described

For ethernet-switching family:

```
<ethernet-switching>
    <port-mode>trunk</port-mode>
    <vlan-id>120</vlan-id>
</ethernet-switching>
```

Where port-mode can be configured as access or trunk.

For inet family, there are some configurable properties such as the address or the MTU.

```
<inet>
    <address></address>
    <mtu></mtu>
</inet>
```

3.2.5 VLANs descriptions

VLANs are described by the JUNOS software as is shown below:

```
<vlans>
    <vlan>
        <name>VLANA</name>
        <description>testing</description>
        <vlan-id>234</vlan-id>
        <interface>
            <name>ge-0/0/0.0</name>
        </interface>
        <interface>
            <name>ge-0/0/19.0</name>
        </interface>
    </vlan>
<vlans>
```

VLAN properties are specified among the <vlan></vlan> tags. Each one of them is described inside the corresponding labels. The more common are:

```
<name> </name>
<description> </description>
<vlan-id></vlan-id>
<interface>
</interface>
```

3.3 VI API

3.3.1 Introduction

The management interface to be used in FEDERICA for virtual nodes (ESXi) is the VMWare Infrastructure API (VI API). The VI API provides a complete set of language-neutral interfaces to the VMware virtual infrastructure management framework. This framework provides an infrastructure for instrumenting, managing and monitoring VMware Infrastructure components, similar to the way that Java Management Extension (JMX) provides an infrastructure for Java applications. Examples of VMware Infrastructure components are virtual machines and host systems, while subsystems, such as a performance manager, are also supported. The virtual infrastructure management framework is accessed by external clients using the VI API.

The VI API is used for the communication between the engine and the virtual machine in the FEDERICA network. Each virtual machine has its own engine which initiates the communication through the VI API. This is similar to the process between a switch engine and a network switch, where a NetConf session is established between the engine and the physical switch.

3.3.2 Language description

VI API is implemented as an industry-standard Web services, hosted on VirtualCenter Server and ESX Server systems. The VI API complies with the Web Services Interoperability Organisation (WS-I) Basic Profile 1.0, which includes XML Schema 1.0, SOAP 1.1, WSDL 1.1.

The Web service provides all the necessary operations, including life-cycle operations, to monitor and manage virtual infrastructure components—compute resources, virtual machines, networks, storage, and the like.

3.3.3 Services

The operations used in FEDERICA to create, adjust and delete virtual nodes are the same as offered in the VMWare Infrastructure SDK. The SDK version used is 1.x, which is different from the 2.0 version. The operations will be mentioned shortly in this section. For further details on the operations and messages involved, we refer to the VMWare Infrastructure SDK Porting Guide, available at www.vmware.com.

Virtual nodes can be created through three operations. These are *Create*, *Clone VM*, or *CreateTemplate*. Parameters in these operations include the configurations for the virtual machine. Once the virtual machine has been created the virtual node can be obtained through the *ResolvePath* operation, while to obtain the resources of the virtual node the *GetContents* method is used.

Virtual nodes can be provisioned with the operation *CreateHardware*. This virtual node can then be updated by the operation *PutUpdates*. Migrating of virtual nodes can be done through two operations. Virtual nodes that are still powered on can be migrated with *MigrateVM*. A virtual node in the powered-off state can be move with the *MoveVM* operation. The first operation is also referred to as hot migration, the latter cold migration.

Two different operations to power on and off a virtual node used for the FEDERICA network are *StartVM* and *StopVM*. Virtual nodes can also be deleted with the Delete operation. The same operation can also be used to delete hosts. If not an entire virtual node but only hosts musts be connected or disconnected, the operations *EnableHost* and *DisableHost* can be used.

3.4 Web Service Description Language

3.4.1 Introduction

In this part of the document the description of the Web Services, implemented to communicate the administrator (or NOC) with both layer 2 and layer 3 resources and also virtual nodes, will be presented. To achieve this, these messages are defined in a SOAP format. The complete list is contained within Web Service Description Language (WSDL) and XML-Schema (XSD) files. This section describes in detail the operations, messages and elements inside these files.

3.4.2 Language description

Web Service Description Language is an XML-based language that provides a model for describing the Web service's public interface. This means that it defines requirements and messages format to interact with the services listed inside its catalogue. WSDL is usually combined with SOAP and XML-Schema: complex data types used are embedded in the WSDL file in the form of XML-Schema. On the other hand, clients should use SOAP to call one of the operations listed in the file.

3.4.3 Namespaces

The table below shows the namespaces referenced in this document:

Prefix	Namespace
wsdl	http://schemas.xmlsoap.org/wsdl/
xsd	http://www.w3.org/2001/XMLSchema
wsa	http://www.w3.org/2005/08/addressing
tns	Reference to the file itself

3.4.4 Switch example

3.4.4.1 Operations

The following WSDL files describe the operations that are available for the NOC to communicate with the switches via Web services. The <portType> tag defines the Web services, the operations that can be performed and the messages needed to call the operations. They are separated in two blocks: factory (shown in the following table) and instance operations for the *EthernetSwitchFactory* and *EthernetSwitch*, respectively. The first ones manage the second ones.

```
<portType name=”EthernetSwitchFactoryPortType”>
 <wsdl:operation name="create">
        <wsdl:input message="tns:CreateInputMessage" />
        <wsdl:output message="tns:CreateOutputMessage" />
        <wsdl:fault name="BaseFault" message="wsrf-bf:BaseFaultMessage" />
 </wsdl:operation>
 <wsdl:operation name="clone">
        <wsdl:input message="tns:CloneInputMessage" />
        <wsdl:output message="tns:CloneOutputMessage" />
        <wsdl:fault name="BaseFault" message="wsrf-bf:BaseFaultMessage" />
 </wsdl:operation>
 <wsdl:operation name="find">
        <wsdl:input message="tns:FindInputMessage" />
        <wsdl:output message="tns:FindOutputMessage" />
        <wsdl:fault name="BaseFault" message="wsrf-bf:BaseFaultMessage" />
 </wsdl:operation>
</portType>
```

Factory operations can create, clone and find instances of *EthernetSwitch* which, in turn, have the operations defined below:

```
<portType name=”EthernetSwitchPortType”>
<wsdl:operation name="invoke">
        <wsdl:input message="tns:InvokeInputMessage" />
        <wsdl:output message="tns:InvokeOutputMessage" />
        <wsdl:fault name="BaseFault" message="wsrf-bf:BaseFaultMessage" />
</wsdl:operation>
<wsdl:operation name="refresh">
        <wsdl:input message="tns:RefreshInputMessage" />
        <wsdl:output message="tns:RefreshOutputMessage" />
        <wsdl:fault name="BaseFault" message="wsrf-bf:BaseFaultMessage" />
</wsdl:operation>
<wsdl:operation name="virtualize">
        <wsdl:input message="tns:VirtualizeInputMessage" />
        <wsdl:output message="tns:VirtualizeOutputMessage" />
        <wsdl:fault name="BaseFault" message="wsrf-bf:BaseFaultMessage" />
</wsdl:operation>
<wsdl:operation name="unvirtualize">
        <wsdl:input message="tns:UnvirtualizeInputMessage" />
        <wsdl:output message="tns:UnvirtualizeOutputMessage" />
        <wsdl:fault name="BaseFault" message="wsrf-bf:BaseFaultMessage" />
</wsdl:operation>
</portType>
```

The invoke operation sends instructions to the device. *Refresh* gets the configuration of the resource and *virtualize* creates a new one, which will be a representation of an interface (port) for the end-users.

3.4.4.2 Messages

The tables in this section show the operation’s message definition inside WSDL files.

EthernetSwitchFactory messages:

```
<wsdl:message name="CreateInputMessage">
        <wsdl:part name="request" element="tns:createReq" />
```

```
</wsdl:message>
<wsdl:message name="CreateOutputMessage">
        <wsdl:part name="response" element="tns:createResp" />
</wsdl:message>
<wsdl:message name="CloneInputMessage">
        <wsdl:part name="request" element="tns:cloneReq" />
</wsdl:message>
<wsdl:message name="CloneOutputMessage">
        <wsdl:part name="response" element="tns:cloneResp" />
</wsdl:message>
<wsdl:message name="FindInputMessage">
        <wsdl:part name="request" element="tns:findReq" />
</wsdl:message>
<wsdl:message name="FindOutputMessage">
        <wsdl:part name="response" element="tns:findResp" />
</wsdl:message>
```

EthernetSwitch messages:

```
<wsdl:message name="InvokeInputMessage">
        <wsdl:part name="request" element="tns:invokeReq" />
</wsdl:message>
<wsdl:message name="InvokeOutputMessage">
        <wsdl:part name="response" element="tns:invokeResp" />
</wsdl:message>
<wsdl:message name="RefreshInputMessage">
        <wsdl:part name="request" element="tns:refreshReq" />
</wsdl:message>
<wsdl:message name="RefreshOutputMessage">
        <wsdl:part name="response" element="tns:refreshResp" />
</wsdl:message>
<wsdl:message name="VirtualizeInputMessage">
        <wsdl:part name="request" element="tns:virtualizeReq" />
</wsdl:message>
<wsdl:message name="VirtualizeOutputMessage">
        <wsdl:part name="response" element="tns:virtualizeResp" />
</wsdl:message>
<wsdl:message name="UnvirtualizeInputMessage">
        <wsdl:part name="request" element="tns:unvirtualizeReq" />
```

```
</wsdl:message>
<wsdl:message name="UnvirtualizeOutputMessage">
        <wsdl:part name="response" element="tns:unvirtualizeResp" />
</wsdl:message>
```

The elements contained in the messages are defined as a XML-Schema. In other words, they are described inside the XSD file which is imported by WSDL.

Factory service:

Create request:

```
<xs:element name="createReq">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="EthernetSwitchData"
            type="tns:EthernetSwitchType">
                        </xs:element>
                        <xs:element name="lifeTime" type="xs:long">
                        </xs:element>
                </xs:sequence>
        </xs:complexType>
</xs:element>
```

createReq: Container element included in the body of a SOAP message when an administrator is sending a create operation to the factory service.

createReq:EthernetSwitchData: Container element inside the create request that defines all the information needed to create the switch representation. EthernetSwitchType will be explained in the next section.

createReq:lifetime: Contains the life time of the new resource in milliseconds.

Create response:

```
<xs:element name="createResp" type="wsa:EndpointReferenceType" />
```

createResp: Contains the Endpoint Reference (EPR) of the switch recently created. The schema of this complex component can be found at the URI presented in the namespace table (section 3.4.3) corresponding with the wsa prefix. Basically, an

Endpoint Reference contains the destination address of a message; in this case the address of the instance Web service.

Clone request:

```
<xs:element name="cloneReq">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="toClone"
                                type="wsa:EndpointReferenceType">
                        </xs:element>
                </xs:sequence>
        </xs:complexType>
</xs:element>
```

cloneReq: Container element included in the SOAP body of a message when a clone operation is sent.

cloneReq:toClone: Contains the EPR to clone.

Clone response:

```
<xs:element name="cloneResp" type="wsa:EndpointReferenceType" />
```

cloneResp: Contains the Endpoint Reference (EPR) of the switch recently cloned responding to an earlier clone request.

Find request:

```
<xs:element name="findReq">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="template"
                                type="tns:EthernetSwitchTemplateType">
                        </xs:element>
                </xs:sequence>
        </xs:complexType>
</xs:element>
```

findReq: Container element included in the body of a SOAP message when an administrator is sending a find operation to the factory service.

findReq:template: Container element inside the find request that defines a template with the information needed to find a switch. EthernetSwitchTemplate will be explained in the next section.

Find response:

```
<xs:element name="findResp">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="resources"
                                type="wsa:EndpointReferenceType"
                                        maxOccurs="unbounded">
                        </xs:element>
                </xs:sequence>
        </xs:complexType>
</xs:element>
```

findResp: Contains the list of Endpoint Reference that matches with the template given in the find request message.

Switch Service:

Invoke request:

```
<xs:element name="invokeReq">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="identifier" type="xs:string">
                        </xs:element>
                        <xs:element name="Parameters"
                                type="tns:ParametersType" nillable="true">
                        </xs:element>
                </xs:sequence>
        </xs:complexType>
</xs:element>
```

invokeReq: Container element included in the body of a SOAP message when an administrator is sending an invoke operation over the switch. This is a generic operation which is used to call the device instructions.

invokeReq:identifier: Contains the identifier of the command, action or operation to invoke. Examples are createVlan or queryResources.

invokeReq:Parameters: The parameters (names and values) that the command, action, or operation needs as an input. ParametersType will be explained in the next section.

Invoke response is empty; no information is needed. Both request and response messages that are included in the refresh operation are empty as well.

Virtualize request:

```
<xs:element name="virtualizeReq">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="Parameters" nillable="true"
                                type="tns:ParametersType">
                        </xs:element>
                </xs:sequence>
        </xs:complexType>
</xs:element>
```

virtualizeReq: Container element for the virtualize operation.

virtualizeReq:Parameters: Parameters needed (names and values) to carry out the operation. It contains, to name a couple of elements, the interface name and the vlan identifier.

Virtualize response:

```
<xs:element name="virtualizeResp" type="wsa:EndpointReferenceType" />
```

virtualizeResp: Contains the EPR of the new resource created responding to an earlier virtualize request.

Unvirtualize request:

```
<xs:element name="unvirtualizeReq" type="wsa:EndpointReferenceType" />
```

unvirtualizeReq: Contains the EPR of the resource to destroy.

Unvirtualize response is empty.

3.4.4.3 Types and Resources

Messages above contain some complex types also defined in the XML-Schema file that will be shown in this section. They describe the manner to get and send the resource information.

EthernetSwitchType: Container element included in a create request message that defines the resource information needed to connect with the device. This type is also used to get the equipment and configuration of the devices.

```
<xs:complexType name="EthernetSwitchType">
        <xs:sequence>
                <xs:element name="Name" type="xs:string" />
                <xs:element name="Model" type="xs:string" />
                <xs:element name="Manufacturer" type="xs:string" />
                <xs:element name="Configuration" type="tns:ConfigurationType" />
                <xs:element name="Equipment" nillable="true"
                        type="tns:EquipmentType" />
        </xs:sequence>
</xs:complexType>
```

EthernetSwitchType:Name: Resource name. The string is the identifier of the Ethernet switch device.

EthernetSwitchType:Model: Contains the model of the device.

EthernetSwitchType:Manufacturer: Contains the switch manufacturer brand.

EthernetSwitchType:Configuration: Container element with all the information necessary to connect with the device: host name, user, password, etc.

EthernetSwitchType:Equipment: Container element with the configuration of the virtual LANs and interfaces.

EthernetSwitchTemplateType: Container element included in a find request message used as a pattern to find a device from its identifier or other parameters.

```
<xs:complexType name="EthernetSwitchTemplateType">
        <xs:sequence>
                <xs:element name="Name" nillable="true" type="xs:string" />
                <xs:element name="Status" nillable="true" type="xs:string" />
        </xs:sequence>
</xs:complexType>
```

EthernetSwitchTemplateType:Name: Name (identifier) of the device to find.

EthernetSwitchTemplateType:Status: Status of the device to find.

ParametersType: Container element used in different operations to insert generic information. It can be used inside invoke messages, because this operation sends different commands to the device.

```
<xs:complexType name="ParametersType">
        <xs:sequence>
                <xs:element name="Parameter" minOccurs="0"
                        maxOccurs="unbounded">
                        <xs:complexType>
                                <xs:attribute name="name" type="xs:string" />
                                <xs:attribute name="value" type="xs:string" />
                                <xs:attribute name="description" type="xs:string"/>
                        </xs:complexType>
                </xs:element>
        </xs:sequence>
</xs:complexType>
```

ParametersType:Parameter:name: Element to identify the parameter.

ParametersType:*Parameter*:*value*: Contains the value of the parameter.

ParametersType:Parameter:description: Parameter's description.

ConfigurationType: Container element included in the *EthernetSwitchType*. It defines the configuration of the device.

```
<xs:complexType name="ConfigurationType">
        <xs:sequence>
                <xs:element name="Module" type="tns:ParametersType">
                </xs:element>
                <xs:element name="Parameters" type="tns:ParametersType">
                </xs:element>
        </xs:sequence>
</xs:complexType>
```

ConfigurationType:Module: Parameters that define the transport and protocol to connect with the Ethernet switch.

ConfigurationType:Parameters: Parameters that define, among other things, the host name, user and password and other security values.

EquipmentType: Container element inside *EthernetSwitchType* with the device configuration.

```
<xs:complexType name="EquipmentType">
        <xs:sequence>
                <xs:element name="VLANInventory" nillable = "true"
```

```
                        type="tns:VLANType" minOccurs="0" maxOccurs="unbounded" />
                <xs:element name="PhysicalInterfacesInventory"
                        type="tns:PhysicalEthernetSwitchPortType"
                        minOccurs="0" maxOccurs="unbounded" />
        </xs:sequence>
</xs:complexType>
```

***EquipmentType:VLANInventory*:** List of virtual LANs configured in the device. This complex type is shown below:

```
<xs:complexType name="VLANType">
        <xs:sequence>
                <xs:element name="VLANName" type="xsd:string"></xs:element>
                <xs:element name="VLANTag" type="xsd:string"></xs:element>
                <xs:element name="VLANDescription" nillable="true"
                        type="xsd:string"></xs:element>
                <xs:element name="Units" nillable="true"
                        type="tns:UnitType" minOccurs="0" maxOccurs="unbounded />
        </xs:sequence>
</xs:complexType>
```

VLANType:VLANName: Contains VLAN name. It must be unique.

VLANType:VLANTag: Contains VLAN id. This is a numeric identifier.

VLANType:VLANDescription: Contains a brief description of the VLAN.

VLANType:Units: List of logical interfaces belonging the VLAN.

EquipmentType:PhysicalInterfacesInventory: List of physical ports configured in the device.

```
<xs:complexType name="PhysicalEthernetSwitchPortType">
        <xs:sequence>
                <xs:element name="Name" type="xs:string" />
                <xs:element name="Description" nillable="true"
                        type="xs:string" />
                <xs:element name="LinkSpeed" nillable="true" type="xs:string" />
                <xs:element name="LinkMode" nillable="true" type="xs:string" />
                <xs:element name="LinkState" nillable="true" type="xs:string" />
                <xs:element name="UnitsInventory" type="tns:UnitType"
                        minOccurs="0" maxOccurs="unbounded" />
        </xs:sequence>
</xs:complexType>
```

PhysicalEthernetSwitchPortType:Name: Physical port's name.

PhysicalEthernetSwitchPortType:Description: Contains a brief description of the port.

PhysicalEthernetSwitchPortType:LinkSpeed: Defines the speed configured in the physical port.

PhysicalEthernetSwitchPortType:LinkMode: Defines the link mode (half-duplex, full-duplex or automatic).

PhysicalEthernetSwitchPortType:LinkState: Defines the state of the physical port (enable, disable).

PhysicalEthernetSwitchPortType:UnitsInventory: List of logical interfaces. In a default configuration, only unit 0 is active.

UnitType: Container element inside both *VLANType* and *PhysicalEthernetSwitchPortType*. It contains logical interface's configuration.

```
<xs:complexType name="UnitType">
        <xs:sequence>
                <xs:element name="UnitName" type="xs:string"/>
                <xs:element name="State" nillable="true" type="xs:string"/>
                <xs:element name="InterfaceName" type="xs:string" />
                <xs:element name="Family" nillable="true" type="xs:string"/>
                <xs:element name="Description" nillable="true"
                        type="xs:string" />
                <xs:element name="IPAddress" nillable="true"
                        type="xs:string" />
                <xs:element name="TrunkPortMode" nillable="true"
                        type="xs:boolean" />
                <xs:element name="VLANs" nillable="true"
                        type="tns:VLANType" minOccurs="0" maxOccurs="unbounded" />
        </xs:sequence>
</xs:complexType>
```

UnitType:UnitName: Contains the name of the logical interface.

UnitType:State: Defines the state of the interface (enable, disable).

UnitType:InterfaceName: Name of the physical port which the logical port belongs to.

UnitType:Family: Interface's family. It can be Ethernet-switching or inet.

UnitType:Description: A brief units description.

UnitType:IPAddress: If family is set as is inet, this is the IP address of the interface.

UnitType:TrunkPortMode: If family is set as Ethernet-switching, this Booleans is true if the port is trunk; false otherwise.

UnitType:VLANs: Contains the list of virtual LANs types in which the logical port belongs to.

3.4.5 Computers (VM servers) example

3.4.5.1 Operations

As mentioned before, *portType* defines the operations that can be performed by the Web services and the messages needed to call them. The same as with switch Web services, these are divided in two blocks: factory (shown in the first table) and instance operations for the computers.

```
<portType name="ComputerFactoryPortType">
        <wsdl:operation name="create">
                <wsdl:input message="tns:CreateInputMessage" />
                <wsdl:output message="tns:CreateOutputMessage" />
                <wsdl:fault name="BaseFault" message="wsrf-bf:BaseFaultMessage" />
        </wsdl:operation>
        <wsdl:operation name="find">
                <wsdl:input message="tns:FindInputMessage" />
                <wsdl:output message="tns:FindOutputMessage" />
                <wsdl:fault name="BaseFault" message="wsrf-bf:BaseFaultMessage" />
        </wsdl:operation>
</portType>
```

Computer factory can create an instance WS and look for an already created instance (find). Computer Web Services offer the services shown below. Some parameters (like fault messages) have been removed in order to introduce the operation clearer. Its messages are shown in the next table.

```
<portType name="ComputerPortType">

        <wsdl:operation name="invoke">
                <wsdl:input message="tns:InvokeInputMessage" />
                <wsdl:output message="tns:InvokeOutputMessage" />
        </wsdl:operation>

</portType>
```

Computer provides invoke operation to do all the actions over computers: get resources (getInventory), create them, or modify its configurations.

3.4.5.2 Messages

All messages of both (factory and instance) services will be presented in this section. Every operation has two messages, request and response:

ComputerFactory messages:

```
<wsdl:message name="CreateInputMessage">
        <wsdl:part name="request" element="tns:createReq" />
</wsdl:message>
<wsdl:message name="CreateOutputMessage">
        <wsdl:part name="response" element="tns:createResp" />
</wsdl:message>

<wsdl:message name="FindInputMessage">
```

```
        <wsdl:part name="request" element="tns:findReq" />
</wsdl:message>
<wsdl:message name="FindOutputMessage">
        <wsdl:part name="response" element="tns:findResp" />
</wsdl:message>
```

Computer messages:

```
<wsdl:message name="InvokeInputMessage">
        <wsdl:part name="request" element="tns:invokeReq" />
</wsdl:message>
<wsdl:message name="InvokeOutputMessage">
        <wsdl:part name="response" element="tns:invokeResp" />
</wsdl:message>
```

Similar to what happens with Ethernet Switch Web Services, elements defined inside messages are described in a XSD file imported by WSDL.

Message elements (createReq/createResp, findReq/findResp, invokeReq/invokeResp) have the same structure here as with Ethernet Switch messages. That is because they have generic parameters. Only the createReq and findReq elements present some differences:

Create request:

```
<xs:element name="createReq">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="Name" type="xs:string" />
                        <xs:element name="Model" type="xs:string" />
                        <xs:element name="Manufacturer" type="xs:string" />
                        <xs:element name="Configuration"
                                type="tns:ConfigurationType" />
                        <xs:element name="Parameters"
                                type="common:ParametersType" />
                </xs:sequence>
        </xs:complexType>
</xs:element>
```

createReq: Container element included in the body of the SOAP message when an administrator is sending a create operation to the factory service.

createReq:Name: Contains name of the device.

createReq:Model: Model of the device: vmwareesxi.

createReq:Manufacturer: Contains the manufacturer of the virtual machine: vmware.

createReq:Configuration: This element contains the type ConfigurationType, explained in the next point.

createReq:Parameters: This element is not used at this moment. It is a way to provide a reservation to possible new parameters or to prevent a change of the platform (xen or virtualbox, for example).

Find request:

```
<xs:element name="findReq">
	<xs:complexType>
		<xs:sequence>
			<xs:element name="Name" type="xs:string" />
			<xs:element name="searchingByKeys" type="xs:boolean" />
		</xs:sequence>
	</xs:complexType>
</xs:element>
```

findReq: Element contained inside find message. It contains:

findReq:Name: Name or key given to find a computer.

findReq:searchingByKeys: If this Boolean is true, *findReq:Name* contains a key identification of the computer. Else, it contains the computer name.

In the next section the types of the resources that are included inside these generic operations are presented.

3.4.5.3 Types and resources

Types used in the computer Web Service are generic and all of them are previously explained in section 3.4.4.3. Specifically, the following types can be used:

ConfigurationType: Contained in the create request messages. Includes information necessary to initiate a computer: IP address, hostname, password and user name.

ParametersType: Generic type composed by a name, value and description. It is used by messages (invokeReq, invokeResp, createReq) and *ConfigurationType* type.

4 Signalling

4.1 Design space and related work

IETF has proposed a document which defines some requirements for new signalling solutions, which conform to the Next Steps In Signalling (NSIS) work group. The main goals of a new solution should be that it should be applicable at a large scale and be lightweight in implementation complexity and resource consumption. Below a short overview is given of the IETF requirements for new signalling solutions. It is recommended to follow these requirements when designing a signalling solution for the FEDERICA infrastructure. Requirements have been divided in obligatory and optional. First the mandatory requirements will be elaborated.

The new protocol must be built up modularly, so advanced features can be made optional. This would make the protocol fit for narrowband solutions. For FEDERICA this is not an important issue, as the protocol would not need to support narrowband networks. However, for compatibility reasons this might be an interesting option to implement. Another requirement is that the signalling protocol must be clearly separated from the information being transported. Also, a new signalling protocol must be independent of the network control mechanism. This way the protocol is able to interoperate with multiple control mechanisms in different networks. Within the FEDERICA-domain this is not an issue, but when connecting to other networks interoperability issues could arise.

The IETF has also defined signalling flow requirements. For example, the placement of signalling senders, forwarders, and receivers must be possible anywhere, and should not be pre-defined. Another signalling flow requirement refers to path coupled or decoupled support. Path coupled signalling, where the signalling information is on the same path as the data, must be supported by a new protocol, due to its QoS advantages.

More requirements can be found in the IETF RFC 3726 document, created by the NSIS working group. Considering the scope of this document, these will not be discussed further in this document. The next section will elaborate on the different signalling taxonomies in existence.

Two signalling approaches can be identified: hard state and soft state [4]. In between these two several hybrid versions exist which could also be applied within FEDERICA. Soft state refers to approaches in which installed states are removed when they time out, unless they are periodically refreshed by a new signalling message. This message should indicate that the state should remain installed. Soft state therefore does not require explicit state removal. Several existing protocols as RSVP and SIP are soft state approaches. Hard state signalling is the exact opposite to soft state signalling. Installed state remains installed unless explicitly removed. This removal is done by a state removal message. If a state installer crashes or leaves

without removing installed state, orphaned states arise; installed states that should have been removed.

The explicit installation and removal used in hard state protocols requires this type to be reliable, where as soft state protocols can implement best effort approaches. Soft state approaches have the advantage that they are not as complex, also leading to shorter convergence times. Several approaches exist which use characteristics from hard state and soft state protocols. For FEDERICA, the type of approach and mechanisms to be applied in the signalling protocol should be decided upon. Below a short overview of existing hybrid approaches.

Soft-state with explicit removal (SS+ER) is the first hybrid approach, and is closest to the soft state approach. It adds an explicit state-removal message to the soft state approach. When state is removed at the sender, the sender transmits a removal message to the receiver.

The second approach is soft-state with reliable trigger message (SS+RT). This adds the reliable transmission of trigger messages, instead of best effort with pure soft state signalling. This is achieved by a trigger acknowledgement sent by the receiver. Another feature is that the receiver sends a message to the sender when a state is removed due to time out. False removal can then be undone by sending a new trigger message.

The third, and last, hybrid approach is the soft-state with reliable trigger and removal message (SS+RTR). This is similar to the previous approach, except that it adds reliable messages (and receiver acknowledgements) for state removal as well.

In total five different approaches can be identified, each with different features. The different features of these approaches could be implemented in the signalling protocol to be designed by UPC for the FEDERICA project, but other features could be added to the protocol.

4.2 Proposed FEDERICA signalling mechanisms

The proposed FEDERICA signalling mechanism is based on the technologies described in chapter 3, NetConf and Web Services. Within FEDERICA the network elements are not configured manually, but through different modules that manage the element. The modules that contain elements are named engines, and each device has its own engine. It also contains some modules that manage the engines. The signalling process is defined as all the communication needed between these two types of modules, with the goal of managing the resources in the network and offering to the user an end-to-end service

Four types of signalling mechanisms can be identified: create, invoke, find and virtualize. These mechanisms will all be described below, explained textually as well as through a sequence diagram.

4.2.1 Create mechanism

At level 2, the signalling messages to create an engine for a switch are *createReq* and *createResp*. The functioning of the create-mechanism is as follows. First, the Network Service sends a *createReq* message to the Factory Web Service, called *ethernetSwitchFactoryService* (ESFS). The Network Service includes switch parameters such as the name, manufacturer, model, host IP address, username, password, transport protocol, etc to the *createReq* message. The Factory Web Service is responsible for creating endpoint references for the resources. Web Services are by nature stateless, as they don't offer possibilities of storing state information. To keep state information, a class Resource will be added by the ESFS. This is done via the *ResourceHome*, which adds the Resource classes. For each physical resource a class Resource is added. The parameters supplied by the Network Service in the *createReq* message are stored in the Resource class. This Resource class is coupled to the *EthernetSwitchService* by the ESFS, to create an endpoint reference. This endpoint reference is the address of the pair *EthernetSwitchService* and Resource, where the Uniform Resource Identifier (URI) of the *EthernetSwitchService* and the *resourceKey* of the resource are used as references.

The Resource class has an attribute Engine, which is capable of creating a new engine for the switch. The method *initEngine* is called to create an engine for the physical resource. This engine establishes a NetConf session with the physical resource. The engine sends a query to the resource requesting the VLAN and interface properties of the resource. The resource then returns the values of these properties to the engine. As the web services are not able to interpret NetConf information, the engine consists of a parser which translates the NetConf data into Java objects.

After this process is finished, the ESFS sends a *createResp* message back to the Network Service. This message contains the end point reference of the specific resource.

4.2.1.1 Sequence Diagram

The sequence diagram for the create process is shown in Fig. 4-1, entering in more detail than the description presented in 4.2.1. The *EthernetSwitchResourceHome* is a component of the *EthernetSwitchService* which actually receives the create petition from the ESFS and forwards this to the *EthernetSwitchResource* class. The *EthernetSwitchResourceHome* then confirms the addition of the resource to the ESFS.

The sequence diagram also enters into more detail on the *initEngine* process. For the moment this process can be done this way, but in the future could be adapted if better solutions are found. A *EthernetSwitchModel* is created which includes the security, transport, protocol, switch ID, model and manufacturer settings for the switch to be added. After this model has been created, an EngineCore is created, on which several queries can be invoked.

The *HibernatePersistenceHelper* is the component which is responsible for translating the Java objects into SQL objects which can be interpreted by the database.

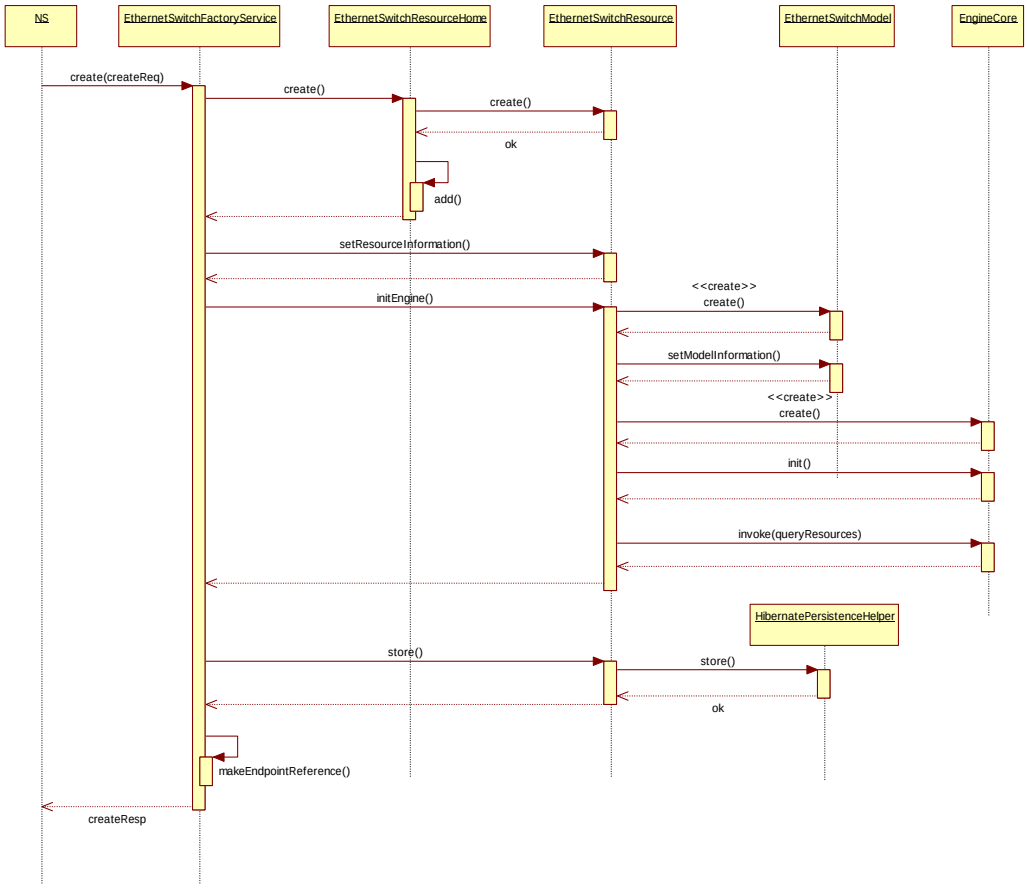

Fig. 4-1. Create Sequence Diagram

4.2.1.2 Example case

This example case describes the process when the network service wants to create an engine for a new switch connected to the network. The switch has the following values for the parameters:

resourceKey	12345678
Version	1.1
Name	UPC1FEDERICA
Manufacturer	Juniper
Model	EX3200
destructionTime	10000
Host IP Address	147.83.118.239
Username	User
Password	Password
Transport	SSH
Protocol	NetConf
Port	22

These parameters can be found in the Resource Database, in the tables: *ethernet_switch_resources*, *switch_configModules*, and *switch_configParameters*. The destruction time is not an important parameter for the moment, and is therefore set to a very high value. In this particular use case, the NOC wants to add a Juniper EX3200 switch, belonging to UPC, to the network. The Network Service sends a *createReq* message to the *ethernetSwitchFactoryService* including the parameters mentioned above in the message body. Upon receiving the *createReq* message, the ESFS then creates a class Resource, naming this ResourceUPC1. All the parameters supplied by the Network Service are stored in this particular class. The ResourceUPC1 creates an engine for the physical switch described in the parameters, with the method initEngine. This engine then establishes a NetConf session and connects to the physical switch. As stated in the parameters, the protocol is NetConf; transport is SSH; and the port used is 22. This means that the engine must connect to port 22 of the physical switch using NetConf over SSH. It knows the IP address of the switch, so it can locate this switch. Access to the switch is gained by introducing the username and password. As soon as the session is established, the engine sends a request for the switch parameters (VLANs, interfaces) to the switch. The switch responds this request and afterwards the session is closed by the engine.

Afterwards, the ESFS creates an endpoint reference for the *EthernetSwitchService* of this switch. This is done by pairing the URI of the *EthernetSwitchService* with the *resourceKey* of the class ResourceUPC1. Since there is only one *EthernetSwitchService* with 1 URI, the value for the URI is the location of the *EthernetSwitchService*. In the FEDERICA case, this would be the location of the

NOC, where all the services are kept. The *resourceKey* is 12345678, as can be seen in the table. Upon successful communication with the switch and creation of the endpoint reference, the ESFS sends a *createResp* message to the Network Service. This message includes the endpoint reference of the physical switch.

4.2.2 Find mechanism

The Find mechanism, initiated by the Network Service, consists of two signalling messages: *findRequest* and *findResponse*. The *findRequest* is sent by the Network Service, the *findResponse* by the *EthernetSwitchFactoryService*. Below, the mechanism is explained in further detail and the sequence diagram is shown in Fig. 4-2.

First, the Network Service sends a *findRequest* message to the *EthernetSwitchFactoryService*. If the Network Service is looking for a specific resource, the name of that resource can be added to the message. The Factory Web Service then requests all the resources from the Resource Database (created by hand currently), communicating through the Resource Service. The Resource Service returns all the resources in the database to the *EthernetSwitchFactoryService*. Depending on if the Network Service requested a specific resource, the results are filtered. The Network Service can only request one specific resource or a full list of all resources. In the latter case, an empty *findRequest* message must be sent. If the *findRequest* message contains a specific resource name, the Factory Web Service is responsible for filtering the resource list for the specific query.

After the filter has been applied, the *EthernetSwitchFactoryService* sends a *findResponse* containing the endpoint reference of the resource requested by the Network Service. In case of an empty *findRequest*, the *EthernetSwitchFactoryService* sends a *findResponse* containing the endpoint references of all resources found in the database.

4.2.2.1 Sequence Diagram

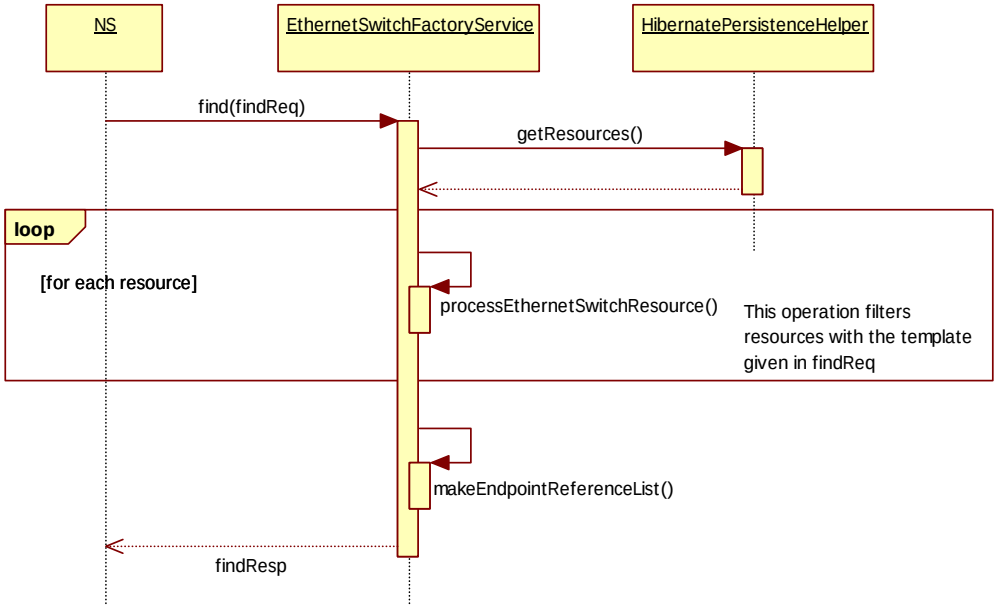

Fig. 4-2. Find Sequence Diagram

The sequence diagram visualizes the process of the find mechanism. The process has been described before in 4.2.2 section. What stands out in this diagram for the text described before is the *HibernatePersistenceHelper*. This element is responsible for retrieving data from the database and parsing this from SQL to Java. After this has been done the ESFS can process the filtering on the search result, make and Endpoint reference list of the coinciding resources and eventually send the response to the Network Service.

4.2.3 Invoke mechanism

The Invoke mechanism is used so the Network Service can make changes to the configuration of a specific resource in the FEDERICA network. It contains the messages *InvokeReq* and *InvokeResp*. The *InvokeReq* can contain several identifiers, depending on the action to be invoked. These actions can affect the logical interfaces, the physical ports, or the VLANs of the resource. Below the mechanism is explained and shown in Fig. 4-3; later a use case will be elaborated.

The first step in the invoke mechanism is initiated by the Network Service. The Network Service communicates with the *EthernetSwitchService*, sending an *InvokeReq* message. This *InvokeReq* contains an identifier, and its parameters, as previously explained in chapter 3. The possible identifiers it can contain are: *createLogicalInterface*, *updateLogicalInterface*, *deleteLogicalInterface*,

configurePhysicalPort, *createVlan*, *deleteVlan*, and *updateVlan*. The *EthernetSwitchService* forwards this request to the engine of the resource for which to invoke the action. The engine parses this request into NetConf, and establishes a NetConf session with the resource. The engine invokes the action on the switch, which applies the action by adapting its configurations. Upon successful application, the switch sends a confirmation message to the engine. The engine closes the NetConf session and sends a confirmation message back to the Network Service.

4.2.3.1 Sequence Diagram

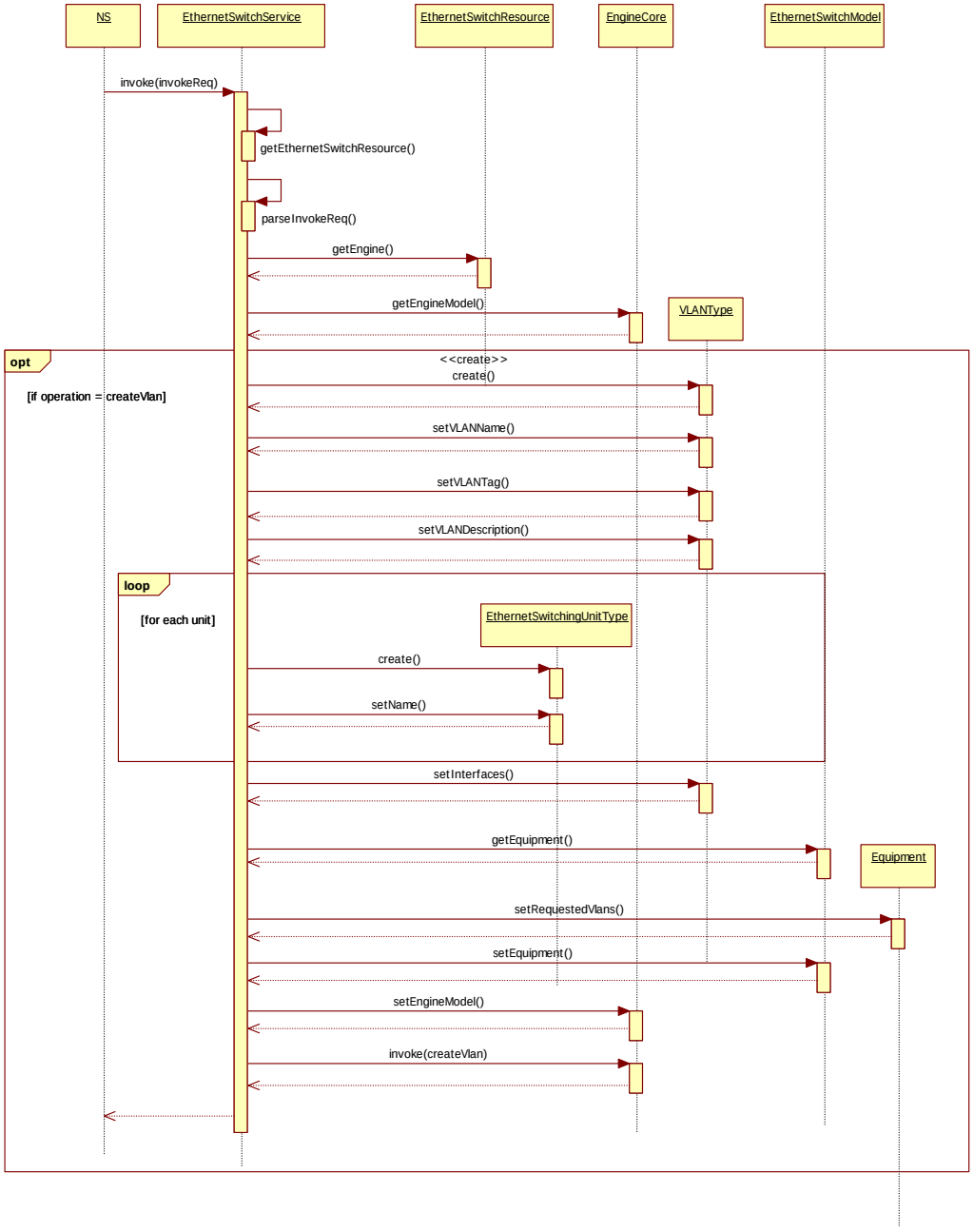

Fig. 4-3. Invoke Sequence Diagram

The sequence diagram shown in Fig. 4-3 gives an example if the createVlan message is invoked by the Network Service. After the resource and engine have been called, the operation to be invoked can be executed. In this figure, a VLAN will be created. The parameters are forwarded from the *EthernetSwitchService* to the VLANType, which is a class within the EngineCore. After this each unit of the switch is created by

calling the *EthernetSwitchingUnitType*. A unit is a virtual interface of the switch, and all units are created in a loop. After the units have been created, the VLANType must be called again to set the interfaces. Equipment is a class of *EthernetSwitchModel* which is the inventory that holds the VLANs and interfaces. The *EthernetSwitchService* has to request the equipment from the *EthernetSwitchModel*, after which it can adjust the *VLANsettings* at the Equipment. These settings are also sent to the *EthernetSwitchModel* in the *setEquipment* message. The *EthernetSwitchService* must also inform the EngineCore of the adjustments, this is done in the *setEngineModel* message. Finally, the Engine is also adjusted with the new VLAN settings. When this process is completely finished, the *EthernetSwitchService* confirms the action to the Network Service, by sending an empty *InvokeReq* message.

4.2.4 Refresh mechanism

The Refresh mechanism is used by the Network Service to update the configuration of a specific resource in the FEDERICA network. It contains the messages *refreshReq* and *refreshResp*. Both usually empty, but can contain extra parameters for extensibility purposes.

The first step in the refresh mechanism is initiated by the Network Service. The Network Service communicates with the *EthernetSwitchService* sending the service operation. The *EthernetSwitchService* forwards this request to the engine of the resource for which to refresh the equipment. The engine establishes a NetConf session and sends the *queryResources* action to the device. Upon successful application, the switch sends a confirmation message to the engine. The engine closes the NetConf session, parses the switch response and updates the equipment with the new configuration. Finally, it sends a confirmation message back to the Network Service.

4.2.5 (Un-)Virtualize mechanism

To virtualize resources, for example a port on a switch, the *Virtualize* mechanism is introduced. It consists of two signalling messages, *VirtualizeReq* and *VirtualizeResp*. These messages have been explained before in chapter 3. This paragraph will explain the functioning of the signalling mechanism.

The virtualize mechanism is initiated by the Network Service, which sends a request to the *EthernetSwitchService*. The *virtualizeReq* message includes the information of the port to be virtualized. The information that must be contained is the contents of the *EthernetResource* table in the database. Upon receiving this message, the *EthernetSwitchService* must request the creation of a new resource by the factory. It does this by sending a *CreateReq* to the Ethernet Resource Factory Web Service. The Factory Web Service then creates a class Resource with all the information received. It also creates an endpoint reference by pairing the *EthernetResourceService* URI with his Resource (WS resource; not FEDERICA's). Upon completion the Resource is

added to the database. A confirmation message must then follow the opposite path to the Network Service.

The *unvirtualize* mechanism follows the exact same communication steps, but instead of a *createReq* message, a destroy operation is applied directly over the endpoint reference provided in the *unvirtualizeReq* message.

4.2.5.1 Sequence Diagram

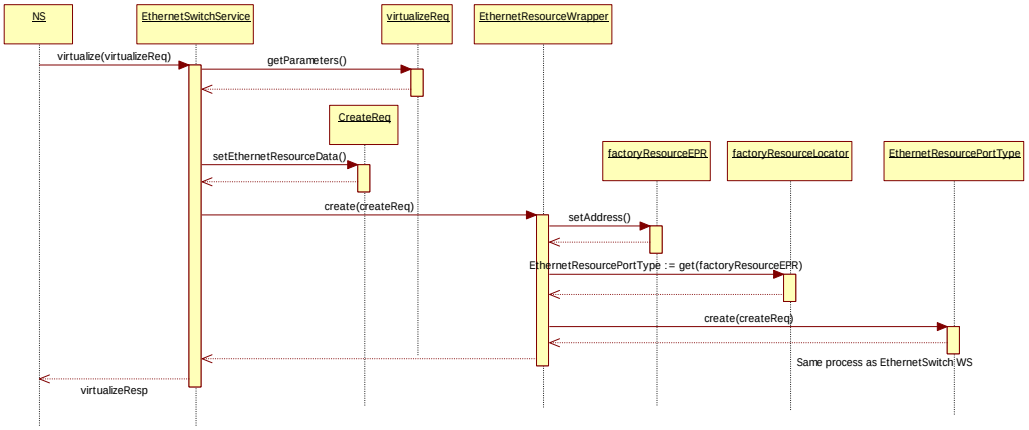

Fig. 4-4. (Un)Virtualize Sequence Diagram

As stated before, the Network Service initiates the virtualization request by sending a message to the *EthernetSwitchService*. All signalling now is mainly concentrated around the *EthernetSwitchService* and the *EthernetResourceWrapper*. This element is responsible for calling the services of the resource to be virtualized. After it has been virtualized a *virtualResp* message is sent from the *EthernetSwitchService* to the Network Service (see Fig. 4-4).

4.3 Resource Discovery

4.3.1 Design Space and related work

Resource discovery covers the location of all resources in the network, resources being nodes, routers, and switches, but also for example services. Within resource discovery several relevant design choices have to be made, when developing a new method. These are listed below, also elaborating on the implications of these choices.

Resource discovery can be either provided as a third party service or as a genuinely distributed system [5]. The first is implemented by a (collection of) server(s) that gather the information on the available resources, and is most common. In a genuinely

distributed system the services are distributed across all resource providers and users, without any central components. A hybrid method would also be possible.

For resource discovery an overlay network is constructed on top of the communication layer. This overlay network can be either manually configured, self-organizing or a hybrid version. With manual configuring a network administrator must design the network graph, which leads to large scalability problems. Self-organizing networks are more qualified for a large network like FEDERICA. However, a good method must be implemented to reduce the network overhead caused by the self-organization of the network. For FEDERICA, it would also be possible to implement a hybrid version, manually configuring the PoPs and using self-organization within the sub-domains. The information that a node must contain before entrance to the system is related to the overlay structure. If manually configured, all nodes need a list of all nodes they interact with. For self-organizing overlay models need no such information (in case of multi- or broadcast), or just the knowledge of one node (in WANs).

The architecture of the overlay network can take on different forms as well, each having different characteristics. All architectures are described as graphs, and these can be trivial, tree, regular, random, or complete. Trivial graphs are centralized systems. Tree graphs are very scalable, but in case of a failure large parts of the network can be affected. Redundancy could be implemented to prevent failures. With regular graphs the nodes are structured orderly. This allows for optimal path computation, but complex algorithms are required to create this model. Random graphs do not have pre-defined links and nodes in the overlay network. Often, these are more robust against failures and offer shorter path lengths, but also increase network overhead and susceptibility to attacks.

References to resource requests must be stored somewhere in the FEDERICA network. This can be achieved in four different manners: local registration only, references, local server registration, or manual registration. With local registration only, only the resource providers are aware of the shared network resources. This is not appropriate for use in the FEDERICA network as it is not scalable, among other disadvantages. If the architecture is a regular graph, references can be used. These are then stored at specific nodes. Registration at local server implies that resource information is stored at the local server (replicated or not). Manual registration is also a possibility, but is not recommended in a dynamic network.

User queries for resources must be routed through the overlay network. For this, two options exist: a central or local replicated server, or query forwarding. In the first case routing is only needed from the user requesting the resource to and from the server. For query routing three possibilities exist: flooding, backtracking, or regular structure routing. Flooding requires limited hop-distance settings (e.g. TTL headers) and loads the network. The main advantage of flooding is the shortest path guarantee. Backtracking is a technique possible in tree structures which recursively and systematically investigates the optimal solution, by rejecting unfit solutions. Regular

structure routing requires a regular graph, and uses predictable locations for the references. User requests must be routed to these locations.

Resource naming is another issue that can be of importance to the development of a resource discovery mechanism. Naming can be based on unique ids or hashes, strings, directory, or attributes. Attribute naming allow for the most extensive queries with predictable results.

For FEDERICA, it is recommended to specify these characteristics of the resource discovery mechanism (and overlay network model) to be developed, based on the requirements of the FEDERICA network. Similar existing technologies and mechanisms can then be studied, broadening the insight on hiatus and development challenges.

4.3.2 Proposed FEDERICA resource discovery mechanisms

In the proposed mechanism, the Network Service is a centralized application/service located at the FEDERICA NOC. One of its functionalities is the discovery of new resources in the FEDERICA network. In this case, it is assumed that only the nodes belonging to the FEDERICA network located at the points of presence are included in the protocol. For the moment it excludes any nodes belonging to the particular subdomains of the institutions connected to the FEDERICA network. The resource discovery protocol consists of three phases; the discovery phase, the control phase and the refresh phase. The function of the first phase is to actively discover new resources in the network. The second phase is then responsible for the control of these resources, in order to maintain a consistent resource database. The refresh phase is responsible for maintaining correct information in the resource database.

4.3.2.1 Discovery Phase

To discover new resources, two possibilities exist. The first is that the Network Service sends an IP broadcast message over the FEDERICA network. As within FEDERICA the points of presence are known, this could be a limited broadcast directed at each point of presence. A node that receives this message and wishes to connect to the FEDERICA network then replies to the Network Service that it wants to connect to the network. However, this has severe complications as the Network Service would have to place large and complex constraints on the search message.

The second option (see Fig. 4-5) is that each resource which wants to connect to the FEDERICA network must transmit a Hello message to the Network Service. Therefore the resource must obtain the address of the NOC Location. This reduces the complexity of the process a lot, since the only requirement is that the resource is aware of the NOC location. Therefore, the proposed mechanism will be developed based on this latter option. Once the Network Service receives this hello message, it

creates an Engine specifically for this switch. This engine establishes a TCP connection with the switch, and communicates with this switch through NetConf. This communication includes configuration mechanisms for the switch and updates the configuration of the switch to connect properly to the FEDERICA network. The switch supplies the engine of information on its available resources, such as ports, VLANs, etc. The engine also parses the resources available in the switch, described initially in NetConf, into objects able to be managed by the program. This engine then transmits these via a Web Service back to the Network Service. The network service stores this information in the Resource Database, which is located at the same location. The logic for the representation is inside the GUI. The Network Service therefore must also send the information to the GUI. A sequence diagram of the discovery phase can be seen below (Fig. 4-5).

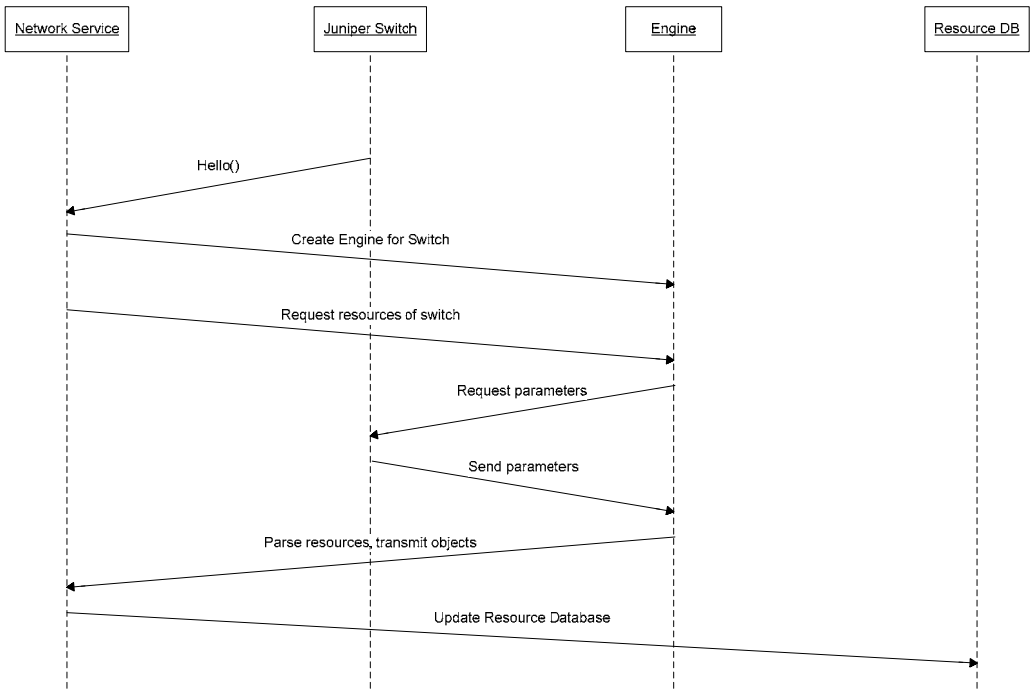

Fig. 4-5. Resource Discovery Phase Sequence Diagram

4.3.2.2 Control Phase

After the resources have been discovered a mechanism must be maintained which ensures correct and consistent information in the Resource Database and in the GUI. In case of a crashing switch, an additional mechanism must be introduced where the engine regularly checks the availability of the switch. The engine transmits a message

which checks the status of the resource. The switch then must reply the Engine that it is still connected and active. In the case that the switch does not reply within the specified time, the engine assumes that the resource has disconnected and notifies the Network Service of this. This then updates the resource database, by removing the objects belonging to that resource from the database. The check status message must repeat itself every certain time period. This value should be configurable by the user. A sequence diagram of this mechanism can be seen below (Fig. 4-6).

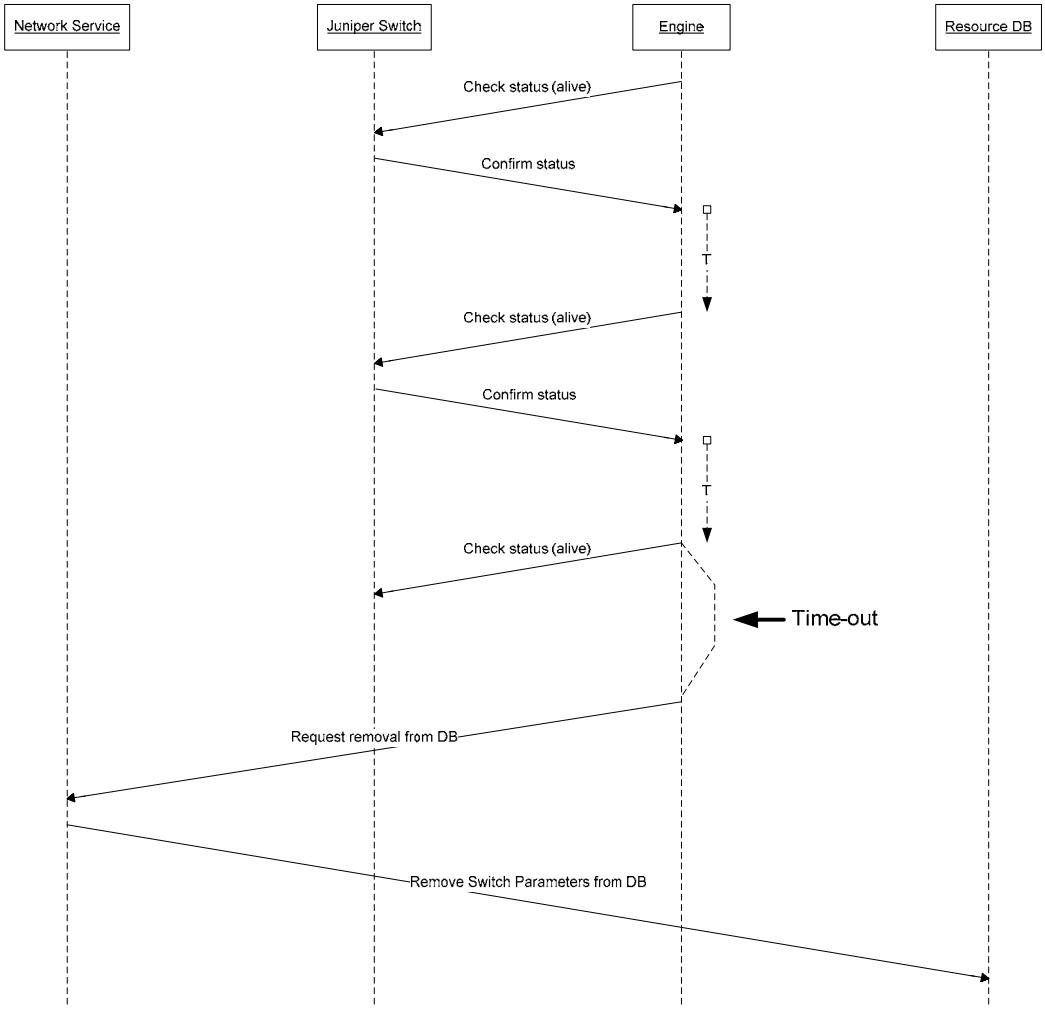

Fig. 4-6. Control Phase Sequence Diagram

4.3.2.3 Refresh Phase

For the refresh phase two possibilities exist, which should be studied further. One option relies on a request from the Network Service for the mechanism to take place. The second option requires a more complex engine which is capable of verifying if the resources available from the switch are the same as already available in the resource database. The engine itself initiates the request and transmits this request to the switch. The switch then replies the engine all its resources. The engine must verify these resources and its values with the previous values. If there are no changes, the mechanism is finished and will repeat itself again after a certain time period. If there are changes in the value, these are parsed and transmitted to the Network Service, which then forwards this to the Resource Database. This option is shown in the figure directly below (Fig. 4-7).

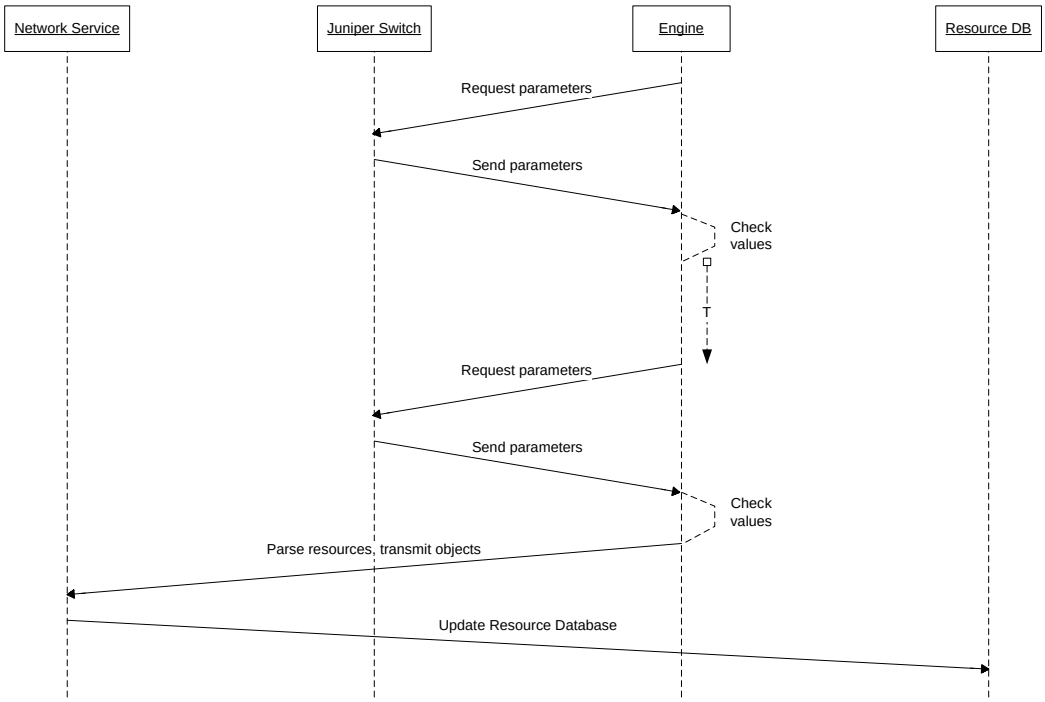

Fig. 4-7. Refresh Phase Sequence Diagram requested by the Engine

The other option is when the Network Service initiates the refresh mechanism, by sending a request to the Engine of the particular switch. The engine requests the resources from the switch, which sends back this information. In this case, the engine is not aware of the previous values of the resource objects, and therefore is not able to verify this data. Therefore all the data must be parsed and sent to the Network Service. This updates the resource database with this data, replacing any objects which might already be included in the database. The sequence diagram for this mechanism is shown directly below (Fig. 4-8).

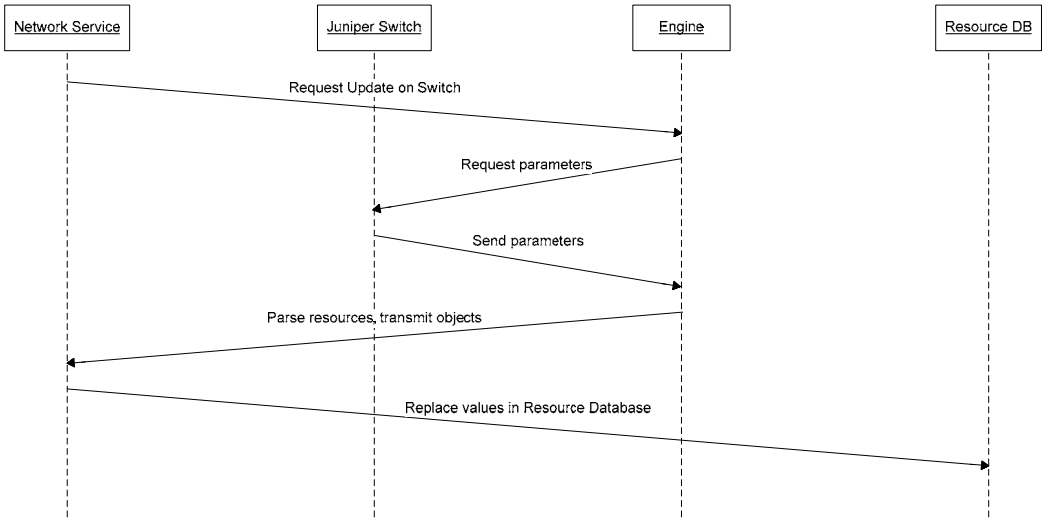

Fig. 4-8. Phase Sequence Diagram requested by the Network Service

4.4 Resource Allocation

4.4.1 Design space and related work

Efficient allocation of physical resources to multiple Virtual Network (VN) requests is considered extremely important since it maximizes the service capacity of the system; the number of coexisting VNs. A potential user of the Federica infrastructure submits a request for a VN consisting of a set of required resources with specific characteristics and attributes (constraints). These requirements form the user's demands from the system (e-infrastructure) for a time interval which may be known in advance or most commonly not. The system in turn must make it possible to satisfy these demands in a cost-efficient way, ideally without compromising other current or future VNs, providing a "slice" of the infrastructure to the user.

Efficient and *fair* utilization of the infrastructure should be the achieved when assigning resources to VN requests. An efficient use of the infrastructure may be considered as the one that maximizes the *acceptance ratio* of VN requests; maximizes the number of coexisting VNs. A fair use of the infrastructure may be considered to be the one that achieves *load balancing* across the infrastructure in terms of utilization of PoPs and links. Although these objectives are quite reasonable for a first approach, there is a lot of space here to appropriately define VN pricing as well as revenue and costs for the system according to the adopted business model [13].

Considering that VN requests are not known in advance, such an assignment of resources to user requests requires the operation of an online algorithm. Contrary to an offline algorithm which is considered to have knowledge of all future VN requests and may make the right decision in assigning resources so as to accommodate the most of them, an online algorithm must make a right assignment decision by not having knowledge of the future. Getting as an input the VN request and the current availability of resources the algorithm should be targeted to satisfy the objectives set and provide the required resources to a VN request, as a slice of the infrastructure. The problem to be solved is also known in the literature as the *virtual network embedding (VNE) problem* [5] which refers to the problem of assigning physical resources to a requested virtual network, with constraints to virtual nodes and virtual links. The VNE problem so far has been faced for assigning nodes and links to a VN request by considering only CPU and bandwidth as the critical resources.

Federica gives the extended capability of providing to a user's request various slices of different types of physical resources (physical elements) such as computing machines, switches, routers and of course links; thus, the problem in our case takes another more complicate form. Attempting an abstraction, we have to deal with the assignment of different types of nodes with each type associated with different, one or more, critical resources to be considered. Each PoP may be considered as a pool of different types of limited resources. A VN request consisting of different types of interconnected virtual nodes with constraints on one (or more) associated critical resources should be efficiently embedded to the e-infrastructure.

The allocation of resources with constraints on virtual nodes and virtual links is known to be an NP-hard problem [7, 8, 9, 10, 11] even in the offline case where all VN requests are known in advance. Since the optimal solution to the problem may not be provided in polynomial time we have to deal with an extremely computationally intensive problem, especially when the number of instances is increased. The development of efficient heuristics is required to provide a good solution in less time.

Considering the online problem makes solution even harder since the properties of incoming VN requests are unpredictable and the developed heuristics must provide a solution within an acceptable time interval from the submission of a VN request. Looking at the system in time, a decision taken at a specific time about a VN embedding may restrict the system's capability of accepting future requests. A very probable case is when a request may not be possible to be satisfied as is, due to lack of resources, but instead it may be possible to be partially satisfied. A partial mapping-assignment of a VN request may take place if the system policy followed permits it while at the same time it is acceptable for the VN request. Such an approach may lead to higher utilizations but it is considered that there are many types of VN requests that it is required to be fully satisfied and a partial assignment is not acceptable. Furthermore, as time passes by, VNs arrive and depart; thus resources are occupied and released respectively. This dynamic process may lead in time in an unbalanced (unfair) use of the substrate network where in some parts may be excessively loaded while in other underutilized. Periodically reconfiguring VNs

would be a solution but interrupting the operation of a VN is not desirable or acceptable in many cases.

4.4.1.1 Practical issues

In order to have a deployable solution this requires having in mind limitations and abilities provided by the hypervisor. The solution to the VN embedding problem for a VN request will provide an assignment of resources to various PoPs. One of our considerations should be if resource allocation is going to be performed manually by taking advice from the output of the algorithm or automatically. There are two approaches in automated resource allocation, one is the coarse approach and the other the fine grained. According to the coarse approach one may consider that resources are already sliced at each PoP and each type of virtual node illustrated belongs to a specific class. Considering virtual machines, a PoP may be ready to provide x "high end", y "normal" and z "low end" virtual machines which will be inactive but ready to be instantly activated. The VN request constraints should be accordingly quantized to the threshold values provided by the system. According to the fine grained approach virtual elements (e.g. virtual machines) are inactive and ready to be activated after being configured "on the fly" according to the resources assigned by the algorithm. For example, if a virtual machine with x MHz of CPU has to be assigned, then before providing one of the ready to be provided virtual machines it is configured appropriately.

In any case, the critical resources to be controlled should be recognized and decided since considering all the resources will lead to an unsolvable problem. In other works in order to relax the problem, CPU is only considered as the critical resource for a virtual machine and bandwidth as the critical resource for a virtual link. Furthermore, the upper bound to the availability of each resource should be considered as well as the units of measure of the availability of each resource.

Another consideration is if reconfiguration of the VN assignment [14] is feasible even for some types of VN assignments. For example, stopping the operation of a virtual machine allocated in one PoP and restarting it to another PoP in order to sustain efficient and fair use of the infrastructure requires further considerations.

4.4.1.2 Previous work

VN assignment shares similarities with older problems such as the embedding of Virtual Private Networks [14], with the difference that in that case there are only bandwidth constraints, as well as with the network testbed mapping problem [15, 11]. In the VN embedding problem there are capacity and placement requirements on both virtual nodes and virtual links while a node and a link is shared by multiple VN requests. This makes the problem hard and even harder in our case where there are *several types of virtual nodes* to be assigned. There are a few works attempting to address the problem more generally, while the majority restricts the problem by studying it in specific VN topologies or considering only the offline case or assuming

infinity capacities or even ignoring one part of the requirements (either for nodes or the links).

In [15] the authors recognizing that all of previous research focused on designing heuristic-based algorithms with clear separation between the node mapping and the link mapping phases provide online VN embedding algorithms with a better coordination between the two phases in order to expand the solution space. They consider CPU as the critical resource of a node and bandwidth as the critical resource of a link. A VN request consists of virtual nodes with CPU constraints and virtual links with bandwidth constraints while at the same time location constraints for each virtual node are also considered. The authors produce an augmented graph (an extension of the physical network graph) including meta-nodes (which are the virtual nodes requested), with meta-links (with unlimited capacity) to those physical nodes that satisfy the constraints. This mapping facilitates their proposed solution. They treat each virtual link with bandwidth constraints as a commodity consisting of a pair of meta-nodes. As a result, finding an optimal flow for the commodity is equivalent to mapping the virtual link in an optimal way.

In [9] the authors focus and provide heuristics on two versions of the VN assignment problem, one without reconfiguration and the other with selective reconfiguration of the most critical VNs. In [15] the authors study the design of the substrate network to enable simpler embedding algorithms and more efficient use of resources, without restricting the problem space. They simplify virtual link embedding by allowing the substrate network to split a virtual link over multiple substrate paths and by employing path migration to periodically re-optimize the utilization of the substrate network.

4.4.2 Resource Allocation in FEDERICA

To be able to come up with a solution to the problem in the Federica case it is required to provide a model for the substrate resources and the virtual network request, accurately define the objectives, recognize and formulate the problem, and finally develop efficient heuristics to provide a good solution in polynomial time.

Solving the problem of VN embedding in Federica and provide a deployable solution requires to specify several different open issues (which is our main current task) while it acquires the synergy of several different tasks such as resource representation, resource discovery and advertisement, resource management. We intend to initially solve the problem in a relaxed but meaningful form while next to consider more complicate cases (e.g. reconfigurations and partial assignments) and elaborate business models.

So far we have attempted to discover an appropriate formulation of the problem in the form it takes in our case. The problem is differentiated and becomes harder compared to previously explored cases since we have to consider *different types of nodes with different associated critical resources*. Our initial intuition indicates that the problem may be formulated as a multi-resource generalized quadratic assignment problem (MR-GQAP) [16] which is a relatively new problem and constitutes a natural generalization of the generalized quadratic assignment problem (GQAP) [17] and the multi-resource generalized assignment problem (MRGAP) [18]. Given *n* jobs, *m* agents, assignment costs of jobs, a cost matrix between jobs, a cost matrix between agents, and coefficients for resource constraints, the objective of MR-GQAP is to find a minimum cost assignment of jobs to agents subject to cardinality constraints (lower and upper bound to the number of jobs assigned to each agent) and multi resource constraints (upper bound to the availability of each different resource available at each agent) for each agent, where the following two types of costs are considered: One is individual cost associated with each assignment, and the other is mutual cost associated with a pair of assignments [16].

MR-GQAP is quite general and one may recognize that our problem shares many similarities with this problem. This becomes easier to comprehend if you think jobs as the virtual nodes to be assigned and agents as the PoPs which provide various resources. Furthermore you may think links in terms of communication cost (bandwidth requirements) between virtual nodes (jobs). We estimate that with an appropriate transformation of the VN embedding problem in Federica it may be well formulated as this known problem. The value of such an approach is obvious since we may benefit of work done for the MR-GQAP which is the extension of the older well studied problems GQAP and MRGAP, and build our extensions and solution over a well studied basis.

Another approach in this resource allocation problem is the usage of the well known genetic algorithm, especially the Particle Swarm Optimization (PSO) method. The multi-objective with constraints (MOPSO) edition has been shown as good candidate in the multiple resource allocation problems in Generalized Quadratic Allocation Problems. We estimate that with a proper formulation of the problem will be able to solve this problem. The possible solution will be valuable in offline planning problems as the online resource allocation seems to be a quite difficult one.

5 Routing

5.1 Design space and related work

The deliverable DJRA1.1 [1] provided conclusions on which tools are appropriate for further investigation. These are BLUEnet Tool, DRAGON, IaaS Framework, and PL-VINI. For routing, BLUEnet tool uses the link state protocol IS-IS. It is unknown in what way extensions of IS-IS are used within BLUEnet tool. In DRAGON, Network Aware Resource Brokers (NARBs) are implemented as agents that represent a single domain. It exchanges information with other NARBs, which present other domains. This enables end-to-end LSP routing. For routing, DRAGON uses a modified version of OSPF-TE. Each LSR must support at least an intra-domain version of GMPLS-OSPF. The NARB acts as a protocol listener in intra-domain routing, and is responsible for inter-domain routing [12].

The IaaS Framework is capable to run over OSPF (link state protocol) and RIP for internal routing and BGP (inter-domain routing protocol); all of them implemented by Manticore. Within PL-VINI, concerning the routing aspects, each PL-VINI node can implement a XORP distribution. XORP implements a number of routing protocols, such as BGP, OSPF, RIP, IGMP, and MLD. There are some issues running different routing protocols simultaneously on the same physical interface.

Any investigation in routing for FEDERICA should preferably be involved with (one of) the routing protocols supported by the tool-benches. All tool-benches but BLUEnet Tool are able to support the OSPF mechanism. This tool can only support IS-IS.

5.2 Proposed FEDERICA mechanisms

5.2.1 Router Bridges

This paragraph gives an overview of the investigation developments regarding routing within FEDERICA. Similar to signalling and resource discovery, only intra-domain issues are mentioned in this paragraph.

Within the IETF, the TRILL work group is developing a new protocol which combines the main features of routing with those of bridging [19, 20]. Within this protocol, RBridges act as hybrid routers/bridges, avoiding most of the limitations presented by current bridged and routed networks. RBridges are backwards compatible with current Ethernet bridges, and routers. As the routing protocol in RBridges a link state protocol was required, narrowing down the choice to either OSPF or IS-IS. The TRILL workgroup has chosen to implement IS-IS as its routing protocol. Implementing RBridges as proposed by the TRILL workgroup within FEDERICA will imply some adjustments within the selected tools, as the remaining tools selected in JRA1.1 all implement OSPF as its main routing protocol. Therefore,

the basis for this research will be to try to design a similar concept to RBridges based on OSPF, instead of being IS-IS based.

The RBridges protocol proposed by the IETF uses IS-IS as its main routing protocol, discarding OSPF. To obtain a further insight in the possibilities of using OSPF within RBridges, the main differences between IS-IS and OSPF will be given here. A complete overview of differences and consequences of these differences can be found in Appendix 1.

Both protocols are link state routing protocols, making them adequate for use with RBridges. The TRILL work group chose IS-IS for two main reasons: IS-IS runs directly at Layer 2, and IS-IS uses a TLV encoding which makes it easy to add new fields to the header. OSPF is not that extendible, but new IETF drafts are discussing the use of TLVs with OSPFv2. OSPF runs on top of Layer 3, resulting in the need that all RBridges must have IP addresses for OSPF to function correctly. This feature makes it difficult to achieve the zero-configuration requirement set by the TRILL working group. Another difference between IS-IS and OSPF are the networks they support. IS-IS cannot support point-to-multipoint networks, which OSPF can. Within IS-IS such a network must be treated as a LAN or as a set of links. Especially the latter solution requires a lot of configuration for these networks. With OSPF both a designated router and a backup designated router are assigned. These will only lose their functions when they drop out of the network. With IS-IS only a designated router is assigned based on its priority, and when a new router with a higher priority enters the network, this will become the new designated router.

Two main features of FEDERICA are the multi-domain aspect and the focus on virtualization. Both of these aspects have not been assessed fully by the TRILL workgroup. The original idea of TRILL is to provide a LAN-oriented solution on a small scale. However, the concept can be translated and designed to work within FEDERICA. This could be between the PoPs only, or even between subdomains connected to different PoPs. This will lead to a completely new implementation of RBridges than currently is being developed, as it involves specific multi-domain issues. As the structure of the subdomains for the networks is unknown, no specific implementations will be discussed for the subdomains. This will give the research a general character, making it possible to apply to other networks as well.

For the implementation of OSPF RBridges several changes in the architecture and the underlying protocols are required. First of all, devices must be added to the architecture which act as routing bridges, in between routers and bridges. These devices should be able to communicate with each other, and also with existing routers and bridges. Because of the OSPF implementation, not only do they need to be assigned a MAC address but also an IP address.

The second adaptation is of the protocol used. Routing bridges do neither communicate via the Ethernet protocol, nor via the OSPF protocol. An encapsulation method for OSPF and Ethernet packets must be designed which makes the communication between routing bridges possible.

5.2.2 Open Flow Implementation

OpenFlow is a forwarding technology developed to isolate flows and to expand the possibilities of existing switching and routing. That is the reason we include it in the routing section.

Nowadays, computer networks are in constant evolution and need to be flexible, scalable and programmable to be ready for future innovations of next generation networks. OpenFlow is a new network architectural platform that intends to simplify network control and management and will make it easier for researchers to test innovative network changes without disturbing existing traffic flows. Using OpenFlow, researchers can experiment with new prototypes for alternative network routing, congestion control or traffic engineering in parallel to a coexisting current platform, by enabling a 'slice' based on flow-level virtualization.

5.2.2.1 Open Standard OpenFlow

The focus of the FEDERICA project lies on providing university researchers with an infrastructure based on virtual environments where they can test new network experiments without running the risk of severely damaging or impacting their current campus networks. An interesting question is now if OpenFlow [21] could support network researchers in a similar way.

The open standard OpenFlow will allow the separation of the data path function and control path function on a switch. In other words, the packet forwarding process of the router still occurs on the router, but high-level routing decisions are moved to a different device, the so-called OpenFlow controller. This controller is typically a standard server connected to the router or switch via a Secure Channel and actually controls the flow table of the router from remote. Communication between controller and OpenFlow switch is defined in the OpenFlow protocol.

Routing of a package can then be accomplished by the OpenFlow router in the following way [22]: Whenever a new packet arrives at the router for which the router cannot find a matching flow entry in its flow table, the router forwards the packet to the OpenFlow controller. It is then up to the controller to make this routing decision, i.e. the controller will decide if the packet should be dropped, or if a new entry must be added to the flow table of the OpenFlow Switch. Once such an entry has been made in the flow table of the router or switch, other packets belonging to that flow can then be forwarded by the OpenFlow router according to the flow table entry for that flow.

Each entry in the flow table consists of three fields: One field consists of the packet header to identify the flow; one field defines the action that is associated with this

flow and thus describes how these packets should be handled; and a third field can be used to collect data on the number of packets in a flow and other useful statistics such as bytes per flow, received errors or the time of the last match for a flow.

In [23], the flow header is a 10-tuple with fields such as ingress port, Ethernet source, Ethernet destination, Ethernet type, VLAN id, IP source, IP destination, IP protocol, TCP/UDP source port and TCP/UDP destination port. A packet is considered a match to a flow table entry if the values in the header fields match the corresponding header field entries in the flow table.

By separating data path and control path between OpenFlow router and OpenFlow controller, the following advantage can be achieved: Researchers can run their new routing protocols or new experiments in real-world traffic settings by separating their traffic into research flows and flows that can be attributed to regular campus operations. Regular traffic flows can then be processed as usual and completely isolated from experimental flows by sending them through the normal processing pipeline of the switch, while experimental flows are subjected to new actions and packet handling. As such, OpenFlow is capable of providing a separate environment for network investigations, which can be compared to FEDERICA's approach of providing a separate environment by using virtual slides for each user.

The following section now presents an overview of the current investigations of OpenFlow within the JRA1 research activity of FEDERICA.

5.2.2.2 Introduction

As part of JRA1, the Friedrich-Alexander University of Erlangen-Nuremberg (FAU) as subcontractor of DFN have started to investigate a hardware implementation of OpenFlow which will lead to experiments with OpenFlow within the FEDERICA slice-based infrastructure.

Initially, the FAU was planning to test the OpenFlow protocol on several switch types, depending on if the appropriate OpenFlow firmware or prototype patches could be obtained. [24] pointed to several vendors for possible switch hardware, such as a Cisco 6509 switch, the Juniper MX-480 switch and possibly a HP ProCurve switch of the 5400 series. All switches are part of the campus network in Erlangen and could therefore be used in such an experiment. An exception is the Juniper MX-480 which is not part of the campus network at FAU, but is part of the FEDERICA test bed and is the DFN core node of FEDERICA and as such also located on the campus in Erlangen.

The FAU’s connection to FEDERICA is currently based on 4 x 1 Gigabit/s. With the support of both traditional campus network environments as well as innovative research infrastructures, the campus network of the University of Erlangen-Nuremberg can serve as a typical representative for university interconnectivity issues. In connection with an OpenFlow infrastructure, interesting focus areas for FAU include the studying of performance and security issues in context with global and

vendor independent access list processing. If possible, the FAU would also like to gain insight into accumulated traffic statistics of flows coming from several switches rather than having only statistics of one single switch at hand.

Fig. 5-1 shows a planned implementation scenario at the University of Erlangen: It involved several Juniper, Cisco and HP switches that would connect the campus network and FEDERICA with traffic being controlled in an OpenFlow experimental implementation.

The planned experiments were in detail:

- Testing the OpenFlow protocol on the FEDERICA infrastructure involving the Juniper MX-480. The experiments were to include the installation of the required Juniper patch to allow OpenFlow processing within a network environment involving a small number of nodes and links. Investigations were planned to aim at performance measurements and security issues.
- The WiN-Labor of DFN at FAU planned to compare performance data of the virtual FEDERICA infrastructure to a real measurements infrastructure, and test the performance of the OpenFlow protocol and other new developments. Measurements of performance data (one way delay, delay variation, packet loss and available bandwidth) were scheduled to be performed with existing tools like HADES (HADES Active Delay Evaluation System) and BWCTL (Bandwidth test Controller).

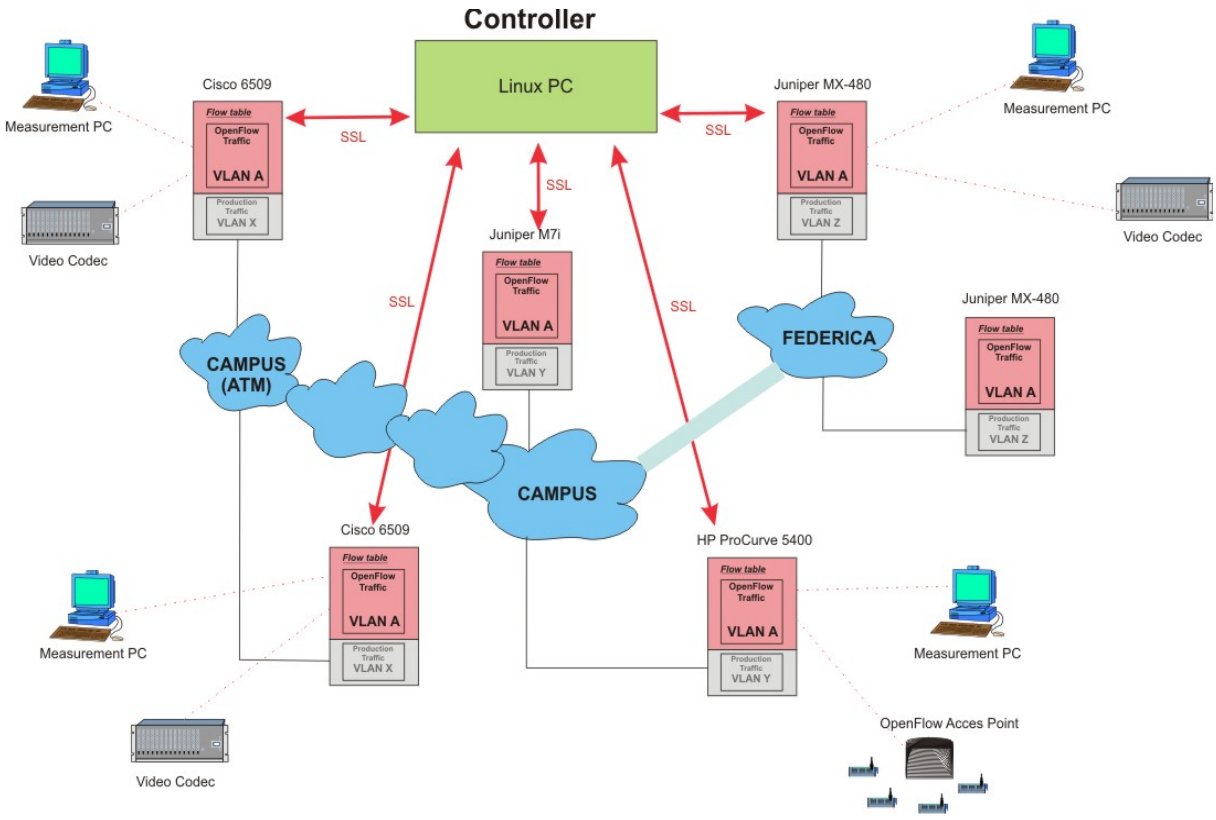

Fig. 5-1. Planned implementation scenario of OpenFlow at the university campus in Erlangen

5.2.2.3 Required Hardware

In order to be able to start on any of these experiments, however, it was necessary to obtain the required hardware patches or specialized firmware for the switches first. At first the FAU used the presentation [24] of Prof. Nick McKeown of Stanford University to identify a list of switch vendors that Stanford had worked with and who already had provided the necessary patches. The listed vendors included Juniper, Cisco, HP and NEC. Although the FAU contacted all of these vendors, the outcome at first was very grim:

- We were informed by Stanford that the Cisco IOS image for the 6500 switches that are used in Stanford are not publicly available.
- We were informed by Juniper that although the OpenFlow implementation was demonstrated at the GENI conference and other places, it was still a prototype version and that OpenFlow was not a GA feature and Juniper had no immediate plans to productize this. We are currently still in contact with

Juniper to see if we could obtain the patch for experimental purposes, but were advised that it would not be possible to use the patch in a complex environment such as a FEDERICA core node, which we initially had intended to do. Furthermore, this patch is under NDA and any publications related to it would be strictly limited by this agreement.

- When we approached the vendor NEC for more information on a possible hardware implementation on an NEC product, our mail was forwarded to their research lab in Heidelberg, but there has not been any reply yet at this time.

- When we approached Hewlett Packard we were informed that there currently are two different patch types available for the HP ProCurve product suite: The patches support OpenFlow version 0.8.9 on the ProCurve 3500 and 5400 series of switches. With the firmware patch the performance is close to 10 Gigabit/s line rate. OpenFlow is also not a supported feature and therefore there is only limited support, but the firmware is available under a software license agreement. We are currently awaiting receipt of the firmware and will then implement it on a ProCurve 3500 switch.

To keep in good contact with the OpenFlow community and all latest developments and news, we joined the email-list openflow-discuss@mailman.stanford.edu as well as the NOX email-list nox-dev@noxrepo.org and hope to keep updated on any changes the vendors may make public through the discussions that can be followed via this list.

Since the hardware implementation has proven to be much more difficult than the literature suggested early on, and since none of the vendors fully support the OpenFlow implementation, it is not clear at this time how much of our planned implementation scenarios and experiments will be able to be carried out.

5.2.2.4 Testing in NOX Environments

In order to be able to experiment with OpenFlow while still waiting for a hardware implementation we started working with the software implementation of OpenFlow and the NOX open-source OpenFlow controller developed by the company Nicira [25]. NOX was created to simplify software development for controlling and monitoring networks, by providing a platform where programs written within NOX (in C++ or Python) can control network forwarding, routing, user / host access and flow control.

In our test we used three workstations where we installed Linux based operating systems (Ubuntu, Debian); one of the workstations was used for hosting both the NOX controller software and the software implementation of OpenFlow for the Linux

kernel. By using *pyswitch*, the Python implementation of a simple ethernet switch for NOX, we controlled the software switch and the two remaining workstations were used as packet generators and analyzers. With this setup it was possible to obtain a proof of concept with a ping test. The simulation was a bit problematic, because there was only very limited documentation that could be found in the NOX sources, the documentation on the NOX homepage is outdated as well and we were not able to add additional filter rules to the controller switch with the *dpctl* tool and its documentation. Because we are still waiting for a hardware implementation of an OpenFlow switch because of the higher performance, we did not investigate the mentioned issues any further.

In another test we conducted an OpenFlow experiment in a completely virtual environment: In this case NOX and the OpenFlow switches as well as end hosts were all supposed to be run and simulated on one and the same host using virtual machines. [26] provides the instructions on how to setup this virtual NOX environment and provides some utility scripts that facilitate the setting up of virtual network topologies. Unfortunately this software deployment was unsuccessful due to a missing debian driver. The work was discontinued when it became clear that indeed a hardware patch from Hewlett Packard would become available.

5.2.2.5 Continued Work

To continue the investigation of a hardware implementation of OpenFlow within the FEDERICA slice-based infrastructure the next step will be to install the hardware patch for the ProCurve 3500 and test its functionalities. After the successful completion of these tests, the test environment can be moved to be integrated into a FEDERICA slice following the FEDERICA user portal at http://194.132.52.194/welcome.do.

6 Monitoring

6.1 Introduction

One of the main goals of the FEDERICA Project is to support research on Future Internet by providing users a pan-European testing infrastructure. The simultaneous access and utilization of that infrastructure is realized by leveraging virtualization capabilities available on networking nodes (Juniper L2/L3 switches) and computing elements (servers).

Virtual environments, named Slices, are built by composing and inter-connecting virtual devices, i.e., virtual networking nodes or virtual computing elements. One of the key aspect for the success of the project consist in being able to efficiently provision, manage and monitor the whole infrastructure (physical and virtual). To this aim, within the project, Joint Research Activities have been defined in order to

produce outcomes on these topics, in such challenging scenario. In this framework our work is aimed at carrying out:

- Analysis of available information and existing software for slice oriented monitoring in FEDERICA (including the work already work in G3).
- Proof of concept aimed to assess the feasibility and highlight the pros and cons of such tool.
- This analysis represents the starting point for the development of a prototype for the FEDERICA Slice-oriented monitoring system.

6.2 (Virtualized) Building Blocks of a FEDERICA Slice

A FEDERICA Slice is a set of resources dedicated to a specific research team. In order to able to assign concurrently many resources to different teams and exploiting the sharing of the FEDERICA physical infrastructure, virtualization capabilities of physical resources are used to build user slices.

Slice building blocks:

- Virtual Switch (implemented by VMWare ESXi software)
- Virtual Machine (and Virtual NICs)
- JUNOS Logical System (Layer 3 virtual router)
- JUNOS Virtual Switches (Layer 2 routing instance)
- VLANs (i.e., logical interfaces of Juniper physical system, logical system or virtual switch)

6.3 Slice Monitoring concept

FEDERICA Slice Monitoring System (FSMS) should provide monitoring of resources within a Slice, that is, monitoring the building blocks of a FEDERICA Slice, i.e., resources listed in the previous section..

In a fully virtualized and heterogeneous environment, as the one implemented in the framework of the FEDERICA project, "flat/plain" monitoring of all available resources, i.e., physical and virtual ones, can lead to confused and poorly intelligible vision of the service provided by the infrastructure, i.e., slices. Furthermore, depending on the virtualization software, specific ad-hoc information and possible limitations on virtual resource monitoring can occur.

In this context, per-Slice monitoring represents a way to ease management and speed troubleshooting of Slices. Moreover, even users can benefit from per-Slice monitoring feature in the cases they have not any available tool for this purpose.

FSMS will be used by:

- FEDERICA NOC for network monitoring and management of the Slice service. The slice monitoring system should expose monitoring data and statistics of resources grouped/organized per Slice
- FEDERICA user. The slice monitoring system will provided users a ready-to-run way to monitor their slices.

6.4 Tool Specifications

Willing features

- Monitoring:
    - Interface status monitor
    - BGP session status
    - Routing protocols status
- Statistics:
    - Weather map
    - Interface (VLAN) traffic and error statistics
    - CPU load and temperature
- Authorization and authentication
    - Per-Slice login credential
- Usability
    - Web based GUI (for data visualization)
    - Easy configuration (by configuration file)
    - Flexibility and open to customization (integration with tools and automated procedures and scripts)

Tool Requirements

- SNMP enabled device monitoring
- Software dependencies, e.g., cron, php, http server, mrtg, etc.

Known limitations

- VMware vSwitches SNMP based monitoring is not available

Specific and general information on virtual resource available in Junos

- Information useful for slice-oriented monitoring will be described here and eventually listed in details in a dedicated annex of the deliverable. In particular, because of the SNMP-based monitoring, detailed information on MIBs will be reported.

Software release

- Package which requires other software and libraries (cron, php, http server, mrtg, etc.). This release can be integrated in the system which is already running the FEDERICA Monitoring System (G3). User access to monitoring data will be provided by a web based GUI.
- Stand-alone virtual appliance including all the software and libraries used by the tools. This release is mainly designed for users interested in customization and integration with other functionalities (e.g., traps, syslog, etc.)

7 Isolation

7.1 Design space and related work

One of the main challenges with experimental network research is to gain sufficient confidence in the experiment results. In general, networks are shared resources with many activities going on at the same time. Those activities influence each other, and the results of an experiment will depend on any other activities going on while the experiment was running.

There are mainly two ways to address this issue. The first is to isolate the experiments so that they run in a dedicated environment, where there is little interference from external factors. However, building large-scale dedicated research networks is expensive and they are not always representative for the type of networks where the applications will be deployed. It is also be possible to organize the network so that different activities are kept separated from each other through means of traffic management, by allocating network resources such as queues, links etc., for the experiments. This, however, requires dedicated support in the network infrastructure and increases the network management workload.

The second way of dealing with this issue is to accept the fact that there are external influences on the experiment results, and look for ways in which those influences can be factored out. This is probably the most common approach, and the way in which an experimental researcher normally deals with this problem is by repeating a measurement many times, and aggregating the measurement value. In that way, the researcher hopes to average out variations that depend on other activities on the network. However, there are several problems with this approach: first, the researcher still cannot be confident that the results are representative, and that the effects of external activities are not visible in the measurement data. Second, this method is time-consuming, and wastes network resources.

Instead, the approach taken here is to use calibration to minimize the systematic and random error in the measurements. Calibration here means to obtain a measure of the amount of external influences on the experiment results, and can be done of almost any variable that might have an effect on the network experiment. For example, it is important to calibrate the computer clocks when doing traffic delay measurements [27].

7.2 Proposed FEDERICA mechanisms

We propose an approach to experimental network research based on experiment calibration. The idea is to perform a calibration measurement in parallel with the experiment. This measurement gives a characterization of the background activities while the experiment is running. Calibration measurements can be compared in order to obtain a measure of the similarity of background activity between different

experiments. If the background activities are similar, then the experiment results are comparable.

The work consists of two main parts: the first deals with development of the methodology, and the second is about design, implementation and evaluation of a tool. In the first part, we will investigate background measurement techniques, and explore how they can be used to gain statistical confidence in the experiment results. Issues that will be investigated here include measurement metrics (throughput, delay, jitter etc.), and measurement methods (such as active and passive measurements). Furthermore, we will study statistical methods for comparing data sets from different background activity measurements. Here we will primarily focus on the Kolmogorov-Smirnov two-sample method, which seems as a promising method for this purpose, and we will also include regular descriptive statistics.

In the second part of the activity, the methodology will be implemented in a calibration tool for FEDERICA, which can be provided to FEDERICA's users as a part of the experiment facilities. The tool will perform background activity measurements, manage background measurement data sets, and analyze and compare data sets with respect to confidence levels.

8 Inter-domain implications

8.1 Introduction

This part of the report will give a first proposition for inter-domain communication. There are two alternatives in this research. In the first alternative, the communication between two FEDERICA slices will be regarded as inter-domain. At the moment the FEDERICA control plane application offers no such functionalities. A research proposition is presented to add these inter-domain functionalities to the FEDERICA network application. This is still under development, so any functionality proposed may vary or be extended in the definite implementation.

One of the main aspects of the interdomain use case in FEDERICA stems from the definition of the term domain. Among the different standardization bodies the term domain is defined differently. Even inside a single standardization body (i.e. IETF) the issue of domain might have different semantics (i.e. DiffServ, BGP, PCE, etc). Recently, this was defined as: *"any collection of network elements within a common sphere of address management or path computational responsibility"* [29][30].

Several proposals for inter-domain communication are proposed in this chapter, varying in functionality and in protocols the proposal is based on. The proposals are presented in more detail in the following sections of this chapter.

In some cases a technology layer could be considered as a domain. In such cases inter-domain would be considered as inter-layer (or cross-layer) functionality, if the objective is to take into account the topology and resource information of these layers. A possible answer on what a FEDERICA technology layer is stems from the definition of Control and Data plane as candidate technology layers.

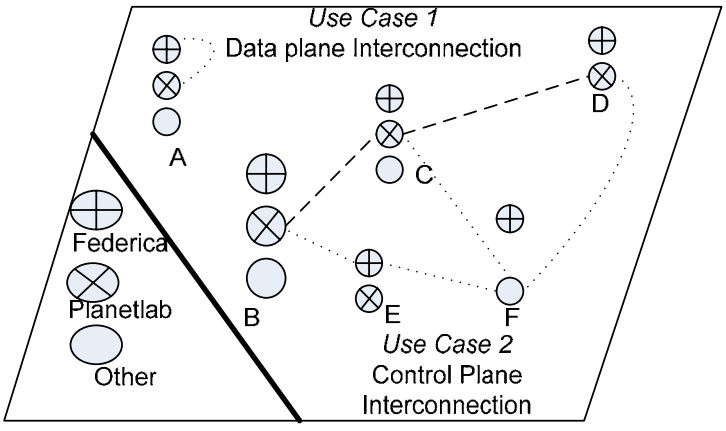

Fig. 8-1. Use case scenarios

In the above figure two possible alternatives are displayed. The first one is the case where there is a request to connect a virtual host (i.e. VM in the VMware world) from Federica to another virtual host from PlanetLab in PoP A. This case is the easiest one and can be considered even in the case of connecting two slices of the same infrastructure. A more difficult case exists when there is an already constructed slice between B, C and D in perforated line and there is a request for a combined link and virtual hosts on different infrastructures at PoPs E and F, connecting them to B and D respectively.

Another proposal takes the Network Service as a basis for the research. In previous chapters the Network Service has been introduced and explained in some detail. This Network Service should be aware of the existing slices, by keeping them in a register (a slice class). This slice class would not only include the resources in the slice, but also some pre-defined inter-slice "communication nodes". Any communication between slices will be done through these "communication nodes". In a first phase, there will be studied how to implement inter-slice communication, working in both Layer 2 and Layer 3. In a second phase, the possible implementation options for inter-slice communication, using only Layer 2, will be studied.

An alternative in this approach is the consideration of inter-domain communication within a hypervisor. Inter-domain communication is a well tested principle and should be examined with respect to layer-2 or layer-3 operation.

Concerning isolation, in an inter-domain situation two possible deployment scenarios exist. First of all, the system can be deployed as a central resource for all users of the network. In the second scenario the system can be deployed individually, in each slice. Individually deployed systems work as described in chapter 7. The measurements taken by a centrally deployed system depends on whether QoS is deployed in the network. If it is not, the centrally deployed system can replace individual deployments, saving the users from running the measurements, as long as the slices are subsets of the physical topology. If not methods such as [28] might provide estimated results from a central measurement system. In a QoS-enabled network, a central deployment can be used to monitor both the health of "empty" slices and the general efficiency of the QoS mechanisms.

The second alternative deals with different FEDERICA physical domains. In this approach we will explore the applicability of ideas and of the Path computation element (PCE) working group of IETF in the area of virtualized resources. The rational for this is based on the fact that a centralized PCE[1] is aware of topology and resources inside a domain. Analogous to this PCE functionality is the Network service functionality. So the PCE communication protocol appears as rational choice.

[1] An entity (component, application, or network node) that is capable of computing a network path or route based on a network graph and applying computational constraints.

8.2 Control plane Interconnection or Network Factory Interconnection

In the control plane of Federica the Network Factory Service provides and controls slices. The Network Factory Service of Federica is a GUI front-end that drives the allocation of resources (links and nodes in general) in various PoPs via a Web service orchestration implemented in SOA. In a SOA populated world the interconnection of network factories is a business process in the control plane which orchestrates data plane directives in a similar way as portrayed in Fig. 8-1. A slightly modified alternative in this direction is the cloud computing approach, where RESTful [33] approach is considered. A RESTful approach burrows its power by a simplistic representation of resources as in WEB 2.0 This scheme, although not generic in use, is a simple yet powerful for the basic user, who can invoke additional resources via API. GoGrid, Amazon Web Service, etc. can be considered as simple use cases driving cloud computing provisioning paradigms. Besides from a restful representation, there is also a need for a functionality which can combine resources.

An alternative approach exists in the generalized network provisioning plane. An extended control – provisioning plane will efficiently manage joint communication and computing resources. In this direction, a variation of G2MPLS **Error! Reference source not found.** developed for concurrent Grid and link resource advertisement / allocation, modified to handle virtualized resources including aspects of resource discovery, resource flooding, signalling etc. could be considered. We coin such variation as VaGMPLS (Virtualization aware GMPLS) and the problem of resource allocation between different virtualization infrastructures can be considered as an Inter-AS Traffic Engineering case adopting automatic switched transport network (ASTN) work on Hierarchical Routing [32].

In this approach and taking into account the consideration of the IETF GMPLS and User Network Interface (UNI) and External Network Node Interface (E-NNI) of Optical Internetworking Forum (OIF) it is possible to develop interoperable procedures for requesting and establishing dynamic resources across heterogeneous networks defined by the automatic switched transport network (ASTN).

Specifically, a modified UNI can be defined as a service control interface between the user devices and the substrate network equipment. It will address the definitions of virtual connectivity services offered by the substrate network, the signalling protocols used to invoke the services, the mechanisms used to transport signaling messages, and the auto-discovery procedures that aid signalling.

On this direction the OIF E-NNI Signalling as an implementation of an open inter-domain signal protocol that enables dynamic setup and release of various services enables end-to-end dynamic establishment of transport connections across multiple control domains.

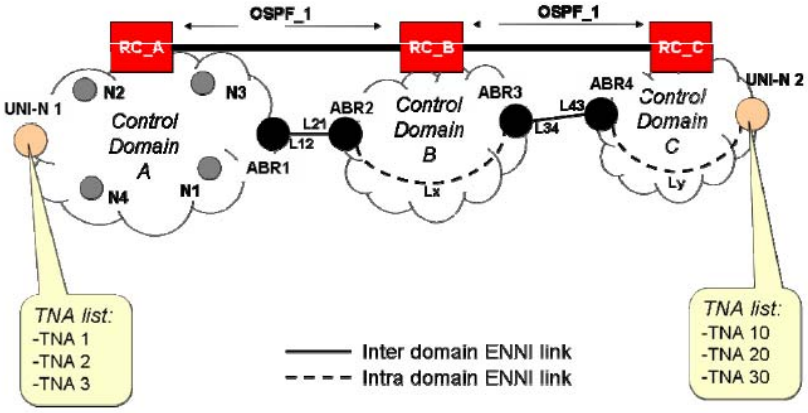

Fig. 8-2. E-NNI hierarchical routing

In each Routing Control Domain (RCD) just one Routing Controller (RC) is defined; the RCD is identified by its RC ID. Neighbouring RCDs are identified through their RC ID and the related (IPv4) address.

E-NNI routing has the main purpose of flooding information about:

- Inter-domain TE links between Area Border Routers (ABR) belonging to different Control bases
- Domains (i.e. OSPF-TE areas)
- General domain capabilities (transit or not, technology, etc.)
- Optionally, summarized intra-domain TE-links, i.e. virtual connections abstractions between Area
- Border Routers (ABR) belonging to the same base Control Domains (i.e. OSPF-TE areas)

In hierarchical routing the info can flow within a hierarchical level upward / downward across routing hierarchical levels. In order to avoid possible advertisement loops, i.e. the info learnt at level N from a higher level (N+1) can be circulated within level N or propagated down to a lower layer. Specific OSPF-TE protocol elements (sub-TLVs) have been defined:

- A Hierarchy List with the RC IDs of the visited levels downward
- An Ancestor RC element in TE link advertisements, specifying the level where the referred link is an intra-domain link: lower, it is a border link, higher than that level it is an intra-domain link.

In E-NNI routing no OSPF multi-area operations are used and the routing information exchanged over the ENNI routing adjacency should only affect the Transport Network topology. This implies that IP-specific LSAs (e.g. Router LSAs, Summary Network LSAs, etc.) are forbidden and just opaque LSAs of type 10 (area-scoped) are flooded.

Unlike traditional OSPF routers that are usually physically adjacent, the E-NNI RCs that would form adjacencies are most likely not topologically adjacent within the control plane. In order to let them create one-hop adjacencies, a variety of methods could be used, such as tunnelling (e.g. GRE, IP-in-IP and IPSec), Layer 2 VLANs, and OSPF virtual links. Each method exposes its own limitations. For example, VLANs can only be applied within SCNs consisting of a single Ethernet broadcast domain; virtual links are an optional capability of OSPF restricted to the OSPF backbone area, etc. OIF E-NNI promotes the use of the OSPF point-to-multipoint method, which consists of creating adjacencies by configuration, by shutting down the automatic OSPF Hello mechanism.

8.3 Path Computation Element (PCE) across multiple domains

In a similar way, the Path Computation Element (PCE) Working Group is chartered to develop a standard Path Computation Element (PCE) based architecture for computation of paths used by MPLS and GMPLS Traffic Engineering. In general the PCE can operate either distributed or centrally depending on the operational principles of the company. PCE Path Computation Clients (PCCs) communicate with Path Computation Elements (PCEs) to address their needs. A PCC can be enabled to dynamically and automatically discover a set of Path PCEs, along with some information that can be used for PCE selection. When the PCE is a Label Switch Router (LSR) participating to the IGP, or even a server participating passively to the IGP, a simple and efficient way for PCE discovery relies on IGP flooding.

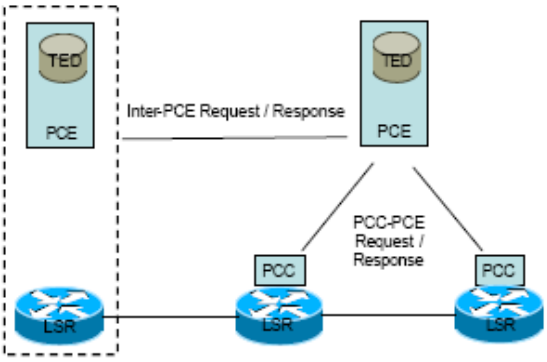

Fig. 8-3. PCE Architecture and PCC-PCE/Inter-PCE Communication

Computed paths could be either explicit PCE paths that list all the intermediate hops, or strict/loose ones that mix specific and abstract hop identifiers, like a router address, or an Autonomous System (AS) number when no detailed topology information is available for confidentiality reasons. The PCE could perform these path computations based on the network graph and the Traffic Engineering Database (TED). The TED could be built by running an IGP with Traffic Engineering extensions, like OSPF-TE or ISIS-TE, or out-of-band via configuration commands. The TED may also include additional information as LSP routes or traffic statistics.

In principle, the availability of the PCE service would enable any TE interdomain operation involving several ASes. In the multi-domain context, it is easier to establish a PCE per AS in order to make interdomain path computation information consistent, facilitate path computation peering agreements and security configuration. In this model, PCEs just need to interact with adjacent PCEs. It appears that the most suitable model for the set up of LSPs spanning multiple domains seems to be *centralized* within domains, and *distributed* (PCE-PCE) on a global basis. If an AS is very large, the domain can be split into regions with communicating PCEs delivering path computation within their regions.

In FEDERICA, a possible PCE implementation would announce not only network resources but also computing and optionally storage resources. This will enable path computations (i.e. LSP) with vector constraints (i.e. computing and networking constraints). Furthermore, end-to-end primary and backup multi-constrained paths may be computed using a distributed loose hop computation approach, in which each PCE along a path could be fed by and use both intra-domain and aggregated interdomain information to compute its corresponding portion of the path.

8.4 Logical Connection based on proposed Network Service

This section will explain in further detail the first alternative offered for interdomain communication, depicting a domain as a slice and interdomain as a logical combination of two or more slices. In order to have a complete sense of the configuration of each slice, a model of each slice is needed. In this model each slice would have to be characterized, including all the resources within the slice. This model acquires special importance considering interdomain implementation. One of the main points to remark in interdomain communication is the point of entrance or exit of each one of the connected domains. This is the resource of the slice, configured to communicate with the other domain edge.

When a slice wishes to connect to another slice, the origin being any node within the slice, the v-node from which the requests originate must contact the Network Service. As explained in previous sections, this communication is achieved through the Engine of the physical resource to which the virtual resource belongs. The Network Service is then responsible for identifying the destination slice and establishing the connection between the slices. For this an additional Class must be added, the SliceClass. This class must contain all the important parameters of the slices in the FEDERICA network, to make inter-communication possible. An example of a parameter of this SliceClass that must be added is the ConnectedToSlice field, which can contain the connections between slices and their configurations. Later, this Class will be described in more detail. Basically, the Network Service must decide a connection point in the destination slice and assign this resource as the interconnection point. By default the node that requests the slice intercommunication will be assigned as interconnection point for the originating slice. This could be adapted, as a virtual

resource could be located in both slices, simplifying the establishment of intercommunication.

Two examples of a possible slice connection are depicted in the figures below. The resources depicted are physical resources, but the figure also includes V-nodes. Although the routers and switches are of course also virtualized, this is not depicted in the figure. The difference between the two figures is whether or not the slices have a physical resource in common or not. This presence will affect the establishment of an inter-domain connection as the unique interconnection points are different in both situations. In the prior case, the Network Service must define the interconnection point. This could be based on proximity to the slice, or other parameters. At the moment this has not been contemplated, but will be in the future. In the second case, with physical resources, the virtual resources which belong to the same physical resource must be assigned the abilities of an interconnection point. This implies that the Network Service must be aware of the physical location of the virtual resources. It must gather this information from the Resource Database, previously explained in section 2.2. The two different options will be described later with use cases.

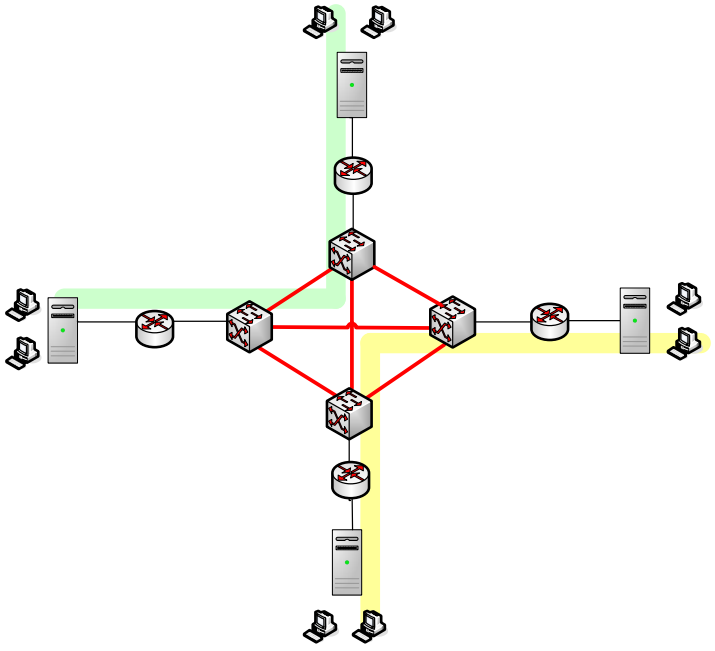

Fig. 8-4. Depiction of physical infrastructure with two slices (separated)

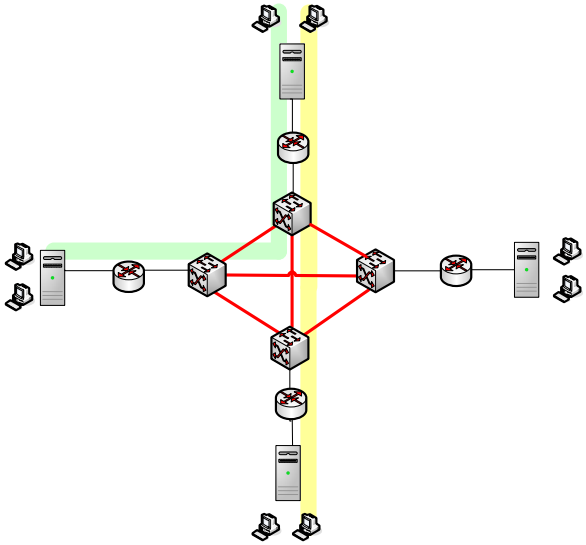

Fig. 8-5. Depiction of physical infrastructure with two slices (connected resources)

In order to establish a communication between two different domains, a specific process will have to be followed. Some of the different steps that compose this process will be have to be done manually, and some others will be done automatically.

The first step to be done is to initiate the process of establishing an interdomain communication. This step can be done by the NOC or by one of the domains. In the first case, the instance will include the resource to be connected to the other domain (the edge resource), and this will be sent to a central element (for example the Network Service) that will send the request to the other domain, which will also have to provide its own edge resource.

In the second case, the central element will have the edge resource information for each one of the domains and start the establishment symmetrically.

After this, the following step in both cases is to establish the communication path between both domains. Two situations are foreseen. The first situation presents two domains without a common physical resource, or with the edge resources in separated physical devices. In this situation, a path will have to be created to join both domains. In the second situation the edge resources of both domains are located in the same physical resource.

In both cases, Q-in-Q may be needed to encapsulate the slice VLAN of each domain adding a second level of labels in the path between both domains. An option would be to use C-VLAN IDs for each logical domain, and adding additional S-VLAN IDs for interdomain communication.

Applying this option, the user would not be able to create his or her own VLANs in a multidomain scenario.

8.5 Data plane Interconnection or slice interconnection

In the Federica environment, the data plane is a slice. The slice is where the end-user resides. So a possible inter-domain case exists when there is a demand of connecting two different virtualization substrates which are based on different technology. For instance, assume the case of connecting Planetlab/Onelab and Federica. The issue of connecting them adheres either in the data-link case where we just want to connect two virtual wires or to connect two VMs. Virtual wire in Federica is an Ethernet segment which could be implemented as a two port Virtualized switch. The interconnection of VMs between Planetlab and Federica adheres to the different technology layer of Virtual ports in the Federica and PlanetLab. A virtual port is a vNIC implemented by host OS (VMWARE in Federica), while this is a tun/tap driver of a UML in the Planetlab. So candidate inter-domain use cases may reside in the data plane as:

- Interconnection of VM between different virtualization substrates
- Interconnection of links

A possible implementation of those connectivity scenarios is shown in the following figure:

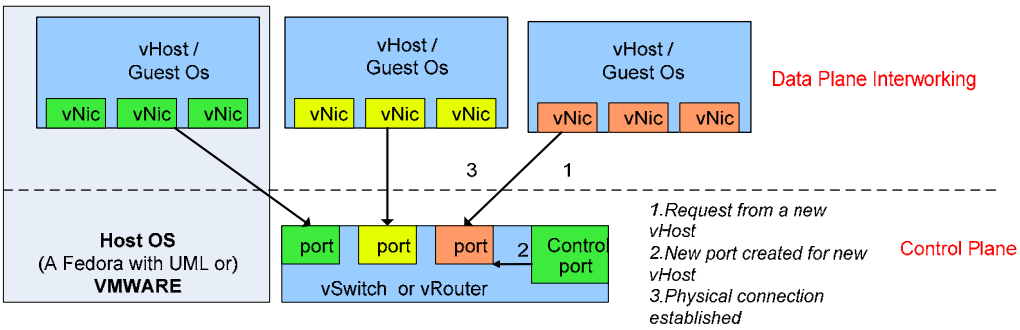

Fig. 8-6. Data plane interconnection between different virtualization substrates

In the above scheme the interconnection is achieved via an auxiliary component such as a vSwitch or a vRouter. In general, a vRouter sometimes requires to be augmented with NAT or Application Level gateway when there is overlap of address space of the connecting slices. The VRouter and/or vSwitch resides in the control plane of the virtualization substrate. The provisioning and operation of this block is beyond the reach of the end user (i.e. slice) similar to that an end user (i.e. application) on Internet can not perceive the operation (i.e. routing) and provisioning of control plane. Furthermore an indicative list of necessary actions in order to achieve the connectivity is displayed.

8.6 Interconnection of virtualized infrastructures in Resource planning.

Resource monitoring remains essential as a preliminary step of any traffic engineering algorithm and is a mandatory step for network design when the resource allocation task is performed. It is typical to employ offline techniques for scheduled requests of combined networking, computation and storage requests. This is referred as Virtual Network Embedding [34], where for a given set of nodes (V) and edges (E) *G(V,E)* (i.e. the virtualization substrate), requests for allocating resources for slices *G'(V',E')* are processed. In general this is an NP hard problem which can be tackled with a more flexible (capabilities) substrate, namely: allowing substrate path splitting and migration.

If the substrate supports mapping a virtual link to multiple physical links/paths, this would essentially relax that constraint. From the user's perspective, he still has one virtual link. But in the substrate, we actually split the packets on the virtual link into a group of substrate paths. Allowing flexible path splitting in the substrate can make it possible to solve the link-embedding problem. Also, with this flexibility, we can get better utilization of the substrate resources than restricting path splitting. In a multidomain environment path splitting could be implemented in a way that links are requested by other neighbour domains thus relaxing the constraints.

The online embedding problem would be much easier to solve if we could migrate an already-running virtual network to another substrate to make room for the new coming ones. In this way, we may attain a more efficient utilization of the substrate resources by accepting more requests. Sometimes, migrating nodes is also necessary for better resource utilization. In general, with ample warning, the migration could be done at a less disruptive and with prior planning.

9 User Portal

9.1 Overview

The procedures for requesting and accessing slice by FEDERICA` users are a subject for investigation, but some general ideas are already available. Thus an effort was taken in order to enable users to use the FEDERICA infrastructure in most convenient and flexible way. The following basic requirements were taken into consideration, as conclusions from several discussions and UPB recommendations:

- User can request of an own private account.
- User can request a new slice by providing information:
  - When the slice should be created
  - When the slice should be removed
  - About slice description
  - About slice topology
  - Upload operating system images files to be started at V-nodes
- User should be informed when the slice is accepted or rejected
- User should be informed when the slice configuration process is completed.
- User should be informed how he can access the created slice.
- User Policy Board Representative can enable, disable or remove User Account
- User Policy Board Representative can approve, reject requested slices
- FEDERICA Network Operating Centre administrator can get information and configure requested slices.

There was decided that the best solution to fulfill all requirements would be a web portal, called the FEDERICA User Portal (FUP). This service is a common web application, which can be accessed from any browser, in respect to FEDERICA security policy. It runs on public IP Address to allow access for all users. Portal prototype has been already deployed at FEDERICA User Access Server.

9.2 Architecture

Main modules of FEDERICA User Portal are shown in Fig. 9-1.

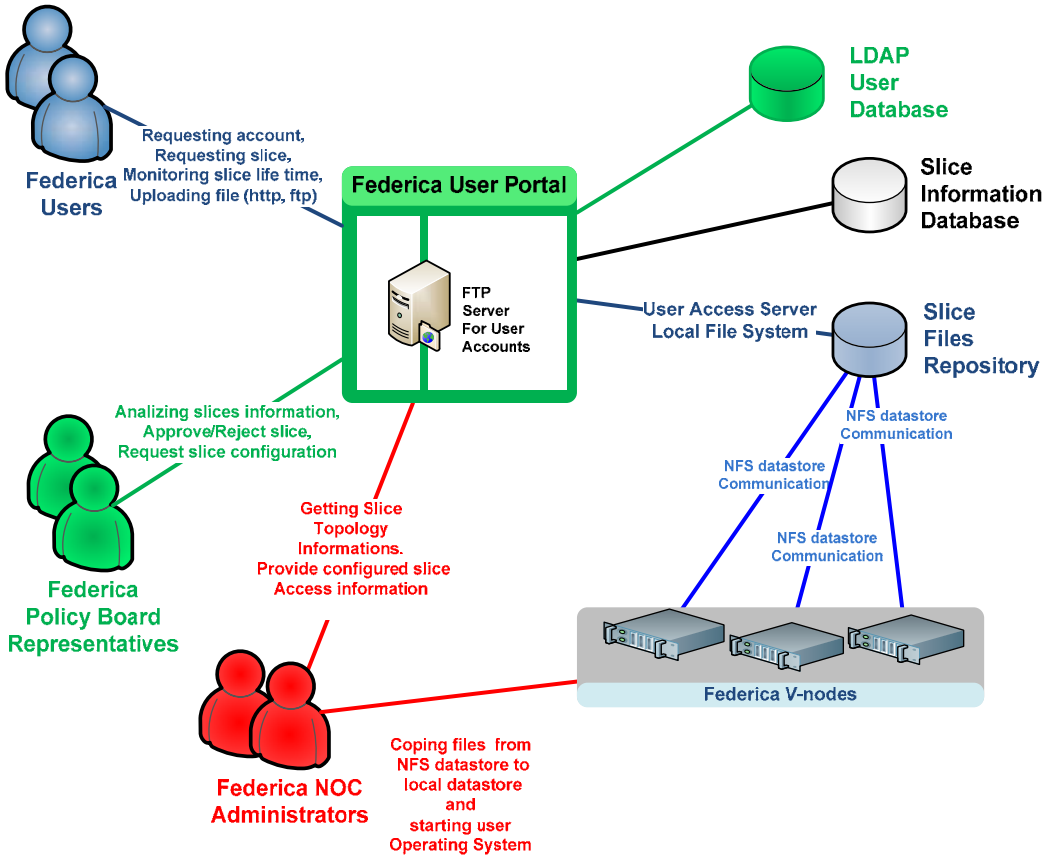

Fig. 9-1. Architecture of the FEDERICA Web Portal

9.2.1 User Database

The User Database should provide users identities for authentication and authorisation in the FEDERICA User Portal or the FTP server. To fulfil these requirements users database was based on a LDAP directory database. The solution provides easy manageable hierarchical structure of AAI information. It allows to group users by country, organisation and to store different authorization attributes. The main advantage of the LDAP directory is possibility to provide access credentials to slices` access servers, with small modification of configuration. User can be authenticated via ssh by using the same password and username provided during registration to FEDRICA User portal. Fig. 9-2 shows the structure of LDAP directory design for the FEDERICA User Portal purposes.

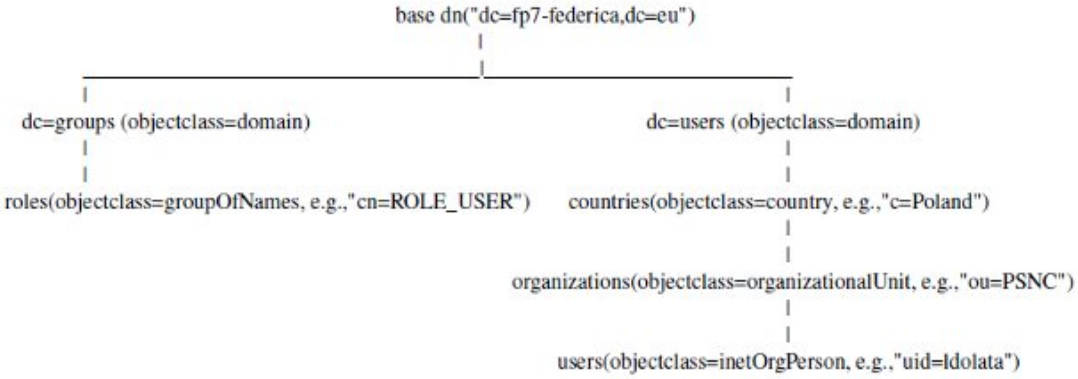

Fig. 9-2. LDAP Directory Schema

The FEDERICA User Portal provides interfaces to add, remove and enable user accounts, without knowing LDAP technology. Each FEDERICA Policy Board Representative can block, at any time access, for users to FEDERICA infrastructure, especially for those who performs negative action against system. This will help to keep FEDERICA environment stable and ready for possible user` attacks.

9.2.2 Slice Information Database

Each FEDERICA user who wants to use the FEDERICA environment in his experiments needs to provide some minimal set of information related to requested slice configuration. These data are stored in dedicated database called Slice Information Database. Information consists of:

- The time when the slice should be configured.
- The time when the slice should be remove.
- Short description of purpose and subject of the experiment.
- The Slice topology information (node, routers, switches configuration and location, connection between elements)
- The Slice files information (virtual appliances files which should be deployed on Vnodes machines)

This information allows a Policy Board Representative to analyse, approve or reject the slice request. After approving slice request, FEDERICA Administrator configures experiment environment and provide configuration information to the user of the slice. All information about slice state change and people involve in approving process also is stored in the Slice Information Database.

9.2.3 Slice files repository

The Slice Files Repository module allows storing slices' files and serving them for V-nodes in FEDERICA infrastructure. It allows user which request a slice to upload needed files by HTTP or FTP protocol. The files are served for V-nodes by Linux Network File System. Each V-node should have configured an NFS repository, to access files from the Slice Files Repository. During slice configuration process administrator needs to copy files from remote NFS data store to local one. After this process the user operating image will be run on the V-node.

9.3 Identified groups of users

Access to the FEDERICA User Portal is restricted only for authorized users. Each user is identified in the system by own private account. This account provides information about personal user data and granted system privileges. It allows a user to log in by typing their user name and password. Authentication and authorization is provided by the FEDERICA LDAP server. After login to the system user can perform new actions and view history of performed ones. List of possible actions depend on a user type. In the first version of the FEDERICA User Portal there are envisaged three kinds of users.

9.3.1 FEDERICA infrastructure User

FEDERICA User is a real user of the FEDERICA Infrastructure. He requests the slice, provides all needed information. After approving process (done by one of User Policy Board Representative) user will use the configured slice` environment slice for his experiments. The slice access information is gained from FEDERICA User Portal.

Each person who wishes to gain access as FEDERICA User to the FEDERICA User Portal needs to provide a registration request. This request is analysed by the User Policy Board Representative and if possible the user account is enabled.

9.3.2 FEDERICA Policy Board Representative

FEDERICA Policy Board Representative is a member of the main decision group in FEDERICA system. This group of users analyses, approves/rejects requested slices and finally requests the slice configuration at NOC or automatic system in the future. This group verifies all slice information and assures that creating slice will not have impact on performance of other slices.

A FEDERICA Policy Board Representative is able to enable or disable user accounts and create account for other FEDERICA Policy Board Representative.

9.3.3 FEDERICA Network Operating Centre Administrator

A FEDERICA Network Operating Centre Administrator is responsible for creating slice in the FEDERICA infrastructure. He allocates all needed resources, copies user files from the Network File System data store to local data store of the V-node. He starts operating systems. After whole configuration process administrator inputs needed information for user to access slice.

10 Conclusions

This deliverable has elaborated further on the research aspects to define the architecture of the control plane for the FEDERICA infrastructure. The focus has been on the intra-domain aspects of the control plane and their properties, and also some inter-domain aspects. The intra-domain aspect has been divided into several topics, which have been researched in order to propose new protocols for the virtual infrastructure. The topics and necessary features covered in this document include resource descriptions, signalling, routing, isolation and monitoring. Another feature that has been studied is the FEDERICA User Portal. Current standardization and existing solutions have been investigated in order to find a good solution for FEDERICA.

The proposal for the control plane introduces a centralized Network Service that will take over some of the logic and be able to automate the tasks of the GUI. To manage the slices, a slice management functionality must be added. The Resource Service needed to support the slices has been elaborated in Chapter 2. The resource description language proposed for FEDERICA is WSDL. Thorough investigation and implementation of this topic has been presented.

Concerning resource allocation, efficient allocation of physical resources to multiple Virtual Network (VN) requests is considered extremely important since it improves the service capacity of the system. FEDERICA gives the extended capability of providing to a user's request various slices of different types of physical resources, making the problem of resource allocation even more complicate in our case compared to other previously studied cases. An optimal solution to the problem may not be provided in polynomial time, thus we have to deal with an extremely computationally intensive problem, especially when the number of instances is increased. The development of efficient heuristics is required to provide online a good solution within an acceptable time interval. In this document we have set the necessary basis in order to be able to come up with an efficient solution to the problem of resource allocation in the FEDERICA case.

As part of the investigation of new routing mechanisms, the possibility and feasibility of a hardware implementation of Open Standard OpenFlow has been researched, which enables 'slices' based on flow-level virtualization. After a thorough market research of switch vendors supporting OpenFlow, a first hardware patch for an HP ProCurve switch could be obtained and installed. The hardware will be used for simulating the virtualization concept of FEDERICA by separating user traffic and making high-level routing decisions via the OpenFlow controller device.

Another investigation on routing has been regarding the possible implementation of RBridges within the FEDERICA network. The proposed protocol would make use of

OSPF instead of IS-IS as the routing protocol. A drawback for this solution is that it requires major hardware changes in the infrastructure.

To summarize, this deliverable lays the foundations for the solutions and protocols proposals for the network control, management and monitoring in a virtualized network context. Next steps will lead to shape these proposals and implement them onto a prototype.

Bibliography

[1] FEDERICA Deliverable SA1.1: "FEDERICA Infrastructure"

[2] FEDERICA Deliverable DJRA1.1: "Evaluation of current network control and management plane for multi-domain network infrastructure".

[3] The Globus Alliance, www.globus.org

[4] P. Ji, Z. Ge, J. Kurose, and D. Towsley, "A Comparison of Hard-State and Soft-State Signaling Protocols", *IEEE/ACM Transactions on Networking*, vol. 15, no. 2, pp. 281-294, Apr. 2007

[5] K. Vanthournout, G. Deconinck, R.Belmans, "A taxonomy for resource discovery", *Personal and Ubiquitous Computing*, Vol. 9, No. 2, pp 81-89, March 2005.

[6] N. M. Mosharaf Kabir Chowdhury, M. R. "Network Virtualization: State of the Art and Reserach Challenges," *IEEE Communications Magazine*, vol. 47, no. 7, pp. 20 – 26, IEEE ComSoc, Jul. 2009.

[7] N. M. Mosharaf Kabir Chowdhury, M. R. "Virtual Network Embedding with Coordinated Node and Link Mapping," *in Proceedings of the 28th IEEE INFOCOM*. Rio de Janeiro, Brazil, pp. 783 – 791, Apr. 2009.

[8] Albrecht, J., Oppenheimer, D., Vahdat, A., and Patterson, "Design and implementation trade-offs for wide-area resource discovery," *ACM Transactions on Internet Technology*. vol. 8, no. 4, pp 1-44, Sep. 2008.

[9] Y. Zhu and M. Ammar, "Algorithms for assigning substrate network resources to virtual network components," in *Proceedings of 25th IEEE INFOCOM*, Barcelona, Spain, pp. 1-12, Apr. 2006.

[10] M. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking virtual network embedding: Substrate support for path splitting and migration," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 17– 29, April 2008.

[11] D. Andersen, "Theoretical approaches to node assignment," Unpublished Manuscript, http://www.cs.cmu.edu/~dga/papers/andersen-assign.ps, 2002.

[12] NARB and RCE Architecture, University of Southern California and Information Sciences Institute, August 2007.

[13] FEDERICA Deliverable DJRA2.1: "Architectures for virtual infrastructures, new internet paradigms and business models".

[14] A. Gupta, J. M. Kleinberg, A. Kumar, R. Rastogi, and B. Yener, "Provisioning a virtual private network: A network design problem for multicommodity flow," in *Proceedings of ACM STOC*, pp. 389–398, Oct. 2001.

[15] R. Ricci, C. Alfeld, and J. Lepreau, "A solver for the network testbed mapping problem," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 2, pp. 65–81, April 2003.

[16] Mutsunori Yagiura, Akira Komiya, Kenya Kojima, Koji Nonobe, Hiroshi Nagamochi, Toshihide Ibaraki, Fred Glover: "A Path Relinking Approach for the Multi-Resource Generalized Quadratic Assignment Problem, " in *SLS* 2007, pp. 121-135. SLS 2007, LNCS 4638, Aug. 2007.

[17] C. Lee , Z. Ma, "The generalized quadratic assignment problem," Technical Report. Department of Mechanical and Industrial Engineering, University of Toronto, Toronto, Ontario, Canada, 2003.

[18] B. Gavish, H. Pirkul, "Algorithms for the multi-resource generalized assignment Problem," *Management Science* 37, 695–713, 1991.

[19] Transparent Interconnection of Lots of Links (trill) http://www.ietf.org/dyn/wg/charter/trill-charter.html .

[20] R. Pelrman et al. "RBridges: Base Protocol Specification", June 2009. draft-ietf-trill-rbridge-protocol-13.txt

[21] N. McKeown et al., "OpenFlow: Enabling Innovation in Campus Networks", available from http://www.openflowswitch.org//documents/openflow-wp-latest.pdf

[22] http://www.openflowswitch.org/wp/learnmore/

[23] B. Heller, OpenFlow Switch Specification v0.8.9, available from http://www.openflowswitch.org/documents/openflow-spec-v0.8.9.pdf

[24] N. McKeown, "Standofr Clean Slate Program", available from http://www.csee.umkc.edu/us-japan-fnw/presentations/mckeown-nsf-nict-us-japan-fnw-08.pdf

[25] http://noxrepo.org/wp/

[26] http://noxrepo.org/manual/vm_environment.html

[27] V. Paxson, "On calibrating measurements of packet transit times," *SIGMETRICS Perform. Eval. Rev.*, vol. 26, pp. 11-21, Mar. 1998.

[28] M. Demirci, S. Lo, S. Seetharaman, and M. Ammar, "Multi-layer Monitoring of Overlay Networks," *International Conference on Passive and Active Network Measurement*, pp. 77-86, April 2009.

[29] RFC 4726, A Framework for Inter-Domain Multiprotocol Label Switching Traffic Engineering, http://www.rfc-archive.org/getrfc.php?rfc=4726

[30] RFC 4655,A Path Computation Element (PCE)-Based Architecture, http://www.rfc-archive.org/getrfc.php?rfc=4655

[31] www.ist-phosphorus.eu/pic/activities/wp1_wp2.pdf

[32] http://www.rfc-editor.org/internet-drafts/draft-ietf-ccamp-gmpls-ason-routing-ospf-9.txt

[33] http://en.wikipedia.org/wiki/Representational_State_Transfer

[34] M. Yu, Y. Yi, J. Rexford, M. Chiang,Flexible Substrate Network to Support Virtual Network Embedding, http://cabernet.cs.princeton.edu/presto07/statements/yu.pdf

Annex 1. IETF Standards

1.1. IETF Standardization Process

The Internet Engineering Task Force is one of the organs that have been active in the standardization process of new technologies. Many thousands of Requests for Comments (RFC) have been published over the years, with some of them ending up being standardized. This section will give a short overview of the standardization process, to explain the difference between an RFC and an Internet Standard.

Each RFC has a status, relative to its step in the Internet standardization process. The RFCs can be divided in standards track and non-standards track documents. An RFC which is not in the standards track can be informational, experimental, or historic. Informational publications are intended to provide general information, which are coordinated adequately to the standards process. Experimental publications are specifications of certain development efforts. Publications that have become obsolete by a newer specification are given the label historic.

A publication in the internet standardization process undergoes the following steps: proposed standard, draft standard, internet standard. A proposed standard is stable, well-understood and is considered valuable within the community. However, it needs further experience before it advances to being a draft standard. As proposed standards may still be modified, implementation of these specifications is only desirable to gain experience or validate the specification. For a proposed standard to become a draft standard, it requires to have at least two independent and interoperable implementations, and sufficient operational experience must be achieved. Draft standards will most likely not undergo any changes, and these can be implemented without problems. A draft standard can become an internet standard, when significant implementation and successful operational experience has been obtained. An internet standard is highly mature and provides significant benefit to the Internet community. Internet standards are assigned STD numbers, alongside their RFC number.

For all IETF documents used as a reference, the type of RFC document has been mentioned behind the RFC number. The topics to be handled by the UPC are resource discovery, routing, signalling, and path computation. In the following sections the existing standards related to these topics will be introduced. The next chapter (chapter 2) will provide the comparison between the functionalities of the tools and the proposed standards.

1.2. Resource Discovery

Regarding resource discovery, neither inter-domain nor intra-domain standards are in existence. Standardization for topology discovery is proprietary. An example of a proprietary standard is the Link Layer Topology Discovery protocol developed by Microsoft. It is included in all Windows Vista versions, and exists also for Linux, but will require a non-free license if it is to be used.

1.3. Routing

1.3.1. OSPF

OSPF is a link state routing protocol, based on hop-by-hop routing communication. OSPF is specifically designed for intra-domain (AS) routing. OSPF, like all link state protocols, requires information about the link cost and has to be able to advertise this link state throughout the network. To calculate the link costs in the network, OSPF uses the Dijkstra algorithm, so the best path through the network can be calculated.

OSPF was the first widely deployed protocol that offered very fast network-wide convergence. One of the main features of OSPF is that it provides the functionality of dividing an intro-domain network into different sub-domains. This allows for a hierarchical setup of the network, which can reduce signalling overhead. Another important feature is that OSPF is capable of supporting several network types; point-to-point, broadcast, non-broadcast multi-access, point-to-multipoint, and virtual links. With OSPF, each router maintains a database that describes the AS topology. This database is used to calculate a routing table by constructing a shortest path three.

OSPF was first put forward as a standard in 1989. Version 2 of OSPF was developed in 1998 (RFC 2328), introducing adjustments to e.g. the flooding mechanism, to external path preferences (to avoid routing loops), and the routing table lookup algorithm. To improve the protocol even more, a third version has also been developed, which supports IPv6 addressing (RFC 5340). A fourth version has yet to be developed, but many extensions to OSPF have been developed within the IETF. RFC 4203 describes encoding of extensions for the support of GMPLS. Traffic engineering support within OSPF was first developed in 1999, in RFC 2676. Further developments regarding TE have been TE extensions for version 3 (RFC 5329), and the support of GMPLS-TE (RFC 5392). The last request for comments also introduces the first support for inter-domain (inter-AS) GMPLS. Also, this RFC includes extensions for inter-domain path computation and flooding between inter-domain links.

IETF documents regarding OSPF:
OSPF version 2 (RFC 2328, STD 54)
OSPF-TE (RFC 2676, experimental)
OSPF extensions in support of GMPLS (RFC 4203, Proposed Standard)

OSPF extensions in support of inter-AS MPLS and GMPLS-TE (RFC 5392, Proposed Standard)
TE Extensions to OSPF Version 3 (RFC 5329, Proposed Standard)
OSPF for IPv6 (RFC 5340, Proposed Standard)

1.3.2 BGP

The Border Gateway Protocol (BGP) is an inter-domain routing protocol. Whereas RIP is a distance vector protocol, BGP is a path vector protocol. The main differences are that BGP avoids looping through path tagging, and that reliable sessions are used for information exchange. Within BGP terminology, autonomous systems are used. These are large network entities, which contain one or more IP-prefix defined networks. All BGP autonomous systems have a unique 16-bit AS number, which is used to determine a path between two autonomous systems. Within each autonomous system, specific entities are BGP speakers, which can communicate with neighbouring autonomous systems. The BGP protocol can also be used as an intra-domain protocol, called IBGP. This allows for two BGP speakers within the same autonomous system to communicate with each other. Currently, BGP is in its fourth version (RFC 4271).

Exchange of network information is done by setting up a communication session between neighbouring autonomous systems. For reliable packet delivery, TCP is used for this communication. This session is used as a virtual link between two autonomous systems. Regarding the virtual links, mechanisms are implemented that offer link preference, ensure link connectivity, and react to link break-downs. The first action after setup of a session is the exchange of all BGP routes. After this the only messages that are exchanged are messages regarding network or path changes.

A critical part of BGP is the route advertisement. This contains specific information about a route at a (set of) IP-prefix network(s), which is called a path attribute. The path attributes are used within the routing decision process. This process consists of two sub-processes; route dissemination and path selection. Both processes are policy based. Route dissemination consists of the export policy and the route aggregation process. Route aggregation is the process of combining IP address blocks from multiple autonomous systems through supernetting. Path selection is based on an import policy, calculating the best route possible.

BGP currently is the accepted standard for Internet routing. Several extensions have been added to BGP. One of the developments has been the support of multiple Network Layer protocols. RFC 4760 describes extensions to carry routing information from IPv6, Layer 3 VPNs, etc. Typically, BGP speakers within a single domain must be fully meshed. This introduces scalability problems, which could be solved using route reflectors, as described in RFC 4456. This technique eliminates the need for full mesh BGP communication, implementing higher-level route reflectors which pass on information to BGP speakers. This can be implemented recursively for larger

networks, to maintain scalability. Currently, the IETF is working on adding an additional attribute to BGP, which allows support of Traffic Engineering. This is still in its draft version, but might be interesting for the FEDERICA infrastructure.

IETF Documents regarding BGP:
A Border Gateway Protocol 4 (RFC 4271, Draft Standard)
BGP-4 Protocol Analysis (RFC 4274, Informational)
Multiprotocol Extensions for BGP-4 (RFC 4760, Draft Standard)
BGP OSPF Interaction (RFC 1403)
BGP Route Reflection (RFC 4456, Draft Standard)
BGP TE Attribute (Draft Version)

1.3.3. IS-IS

IS-IS, which stands for Intermediate System to Intermediate System, is an interior routing protocol that is defined in ISO/IEC 10589:2002 as an OSI standard. IS-IS did not support the Internet Protocol, which the IETF developed OSPF did. An extension has been made to support IP, which is called Integrated IS-IS and is defined in RFC 1195.
Like OSPF, IS-IS also provides a network hierarchy using different areas. The main difference is that with OSPF a router can be placed on the edge of an area, while with IS-IS connections between areas are only through links. IS-IS is also capable of support different network types, such as broadcast and point-to-point networks. An advantage of IS-IS over OSPF is that IS-IS runs directly over Layer 2 links, as IS-IS packets are encapsulated in Layer 2 frames. This means that IS-IS is more secure than OSPF.
IS-IS has been undergoing some developments in 2008, with several extensions being proposed in RFCs. Support of Traffic Engineering is described in RFC 5305, by specifying new types of information that a router can implement. Support of GMPLS has been described in RFC 5307, and the support of IPv6 is described in RFC5308. For this, two new Type-Length-Values have been added to the protocol.

IETF Documents regarding IS-IS:
Use of OSI IS-IS for Routing in TCP/IP and Dual Environments (RFC 1195, Proposed Standard)
IGP Routing Protocol Extensions for Discovery of Traffic Engineering Node Capabilities (RFC 5073, Proposed Standard)
IS-IS Extensions for TE (RFC 5305, Proposed Standard)
IS-IS Extensions in support of GMPLS (RFC 5307, Proposed Standard)
Routing IPv6 with IS-IS (RFC 5308, Proposed Standard)
IPv6 TE with IS-IS (Draft, as of January 2009)

1.3.4. RIP

Routing Information Protocol, or RIP, was the first protocol used in an intra-domain environment for IP networks. RIP exists in two versions, described in RFC 1058 and RFC 2453, where version 2 extends the first version in several ways. With RIPv1 communication of routing information is always done between two neighbouring routes. As RIPv1 is a distance vector protocol, distance vector information must always be obtained from the neighbouring router. The major drawback of RIP is that the protocol is UDP-based. Since UDP does not guarantee delivery, there is no guarantee that a RIP message is received by a router. Another drawback of RIP is the limited scalability, as the destination cannot be further than 15 hops away. RIPv1 would therefore be good to use in small networks, but does not seem adequate for use in FEDERICA.

RIPv2 extends some of the capabilities of RIPv1. RIPv2 allows for subnet masking and introduces authentication to the protocol. For the subnet masking, variable length subnet masking is supported. To support authentication a first entry block of 20 Bytes can be allocated. Authentication does limit the routing capabilities of a message, as authentication uses one of the twenty-five routing table entries. Another development of RIPv2 is the support of IPv6, described in RFC 2080.

IETF Documents regarding RIP:
RIPng for IPv6 (RFC 2080, Proposed Standard)
RIP Version 2 (RFC 2453, STD 56)
RIP Version 2 Protocol Applicability Statement (RFC 1722, STD 57)

1.4. Signalling

Two main signalling protocols have been under development by the IETF. These are constraint-based LSP setup using LDP (CR-LDP), and Resource Reservation Protocol (RSVP). RFC3468 describes the decision of the IETF to undertake no further action on the development of CR-LDP and focus all its activities on development of the RSVP-TE protocol. However, a short overview of CR-LDP and its RFCs will be given in this document.

CR-LDP contains extensions for the prior Label Distribution Protocol (described in RFC 3036). It introduces the ability to LDP to setup Label Switched Paths based on explicit constraints, such as QoS constraints or route constraints. Because of these developments, Traffic Engineering requirements are supported by the CR-LDP. CR-LDP is described in RFC 3212, and the applicability of CR-LDP has been described in RFC 3213. RFC3214 describes the functionalities of modifying several parameters of a pre-established CR-LSP using CR-LDP. This is to support varying requirements, such as bandwidth reservation, etc. As CR-LDP has been discarded for further development by the IETF, this document will not go into further detail of the existing RFCs. It has been removed from the IETF Standard Track, and therefore research on CR-LDP developments is not interesting within the FEDERICA project.

The RSVP protocol (RFC 2205) is used by a host to request specific qualities of service from the network for particular application data streams or flows. RSVP is also used by routers to deliver QoS requests to all nodes along the path(s) of the flows and to establish and maintain state to provide the requested service. RSVP requests will generally result in resources being reserved in each node along the data path. RSVP defines how applications place reservations and how they can release the reserved resources once the need for them has ended. RSVP operation will generally result in resources being reserved in each node along a path. RSVP was designed to interoperate with current and future routing protocols.

The RSVP-Traffic Engineering Protocol (RFC 3209) is an extension of the RSVP protocol for traffic engineering. It is used as a general facility for creating and maintaining distributed forwarding and reservation state across a mesh of delivery paths. Applications running on IP end systems can use RSVP to indicate to other nodes the nature (bandwidth, jitter, maximum burst, etc.) of the packet streams they want to receive. The extended RSVP protocol supports the instantiation of explicitly routed LSPs, with or without resource reservations. It also supports smooth rerouting of LSPs, pre-emption, and loop detection. The LSPs created with RSVP can be used to carry "Traffic Trunks". The LSP that carries a traffic trunk and the traffic trunk by itself are different but related concepts. For example, two LSPs between the same source and destination could be load shared to carry a single traffic trunk. LSPs can be treated as tunnels, because the traffic that flows along one of these LSPs is defined by the label applied at the ingress node of the LSP. When an LSP is used in this way we refer to it as an LSP tunnel. LSP tunnels allow the implementation of different network performance optimization policies. LSP tunnels can be automatically or manually routed away from network failures, congestion, and bottlenecks. Multiple parallel LSP tunnels can be established between two nodes, and traffic between the two nodes can be mapped onto the LSP tunnels according to local policy.

Several extensions to RSVP-TE have been proposed as standards. RFC 3473 describes the extensions to RSVP-TE to support GMPLS. Specific formats and mechanisms are introduced to support the GMPLS classes of interface and switches. Regarding GMPLS, RFC 5151 describes the extensions for RSVP-TE needed for inter-domain MPLS and GMPLS. The main feature introduced is the establishment and maintenance of LSPs that cross domain boundaries. Last, RFC 4872 introduces extensions to GMPLS RSVP-TE for recovery (protection and restoration) of end-to-end LSPs.

For point to multipoint links one standard has been proposed. RFC 4875 describes a solution which does not require a multicast routing protocol in the SP core network, and is able to set up point to multipoint LSPs in GMPLS networks. This does not include inter-domain solutions, a document for inter-domain point to multipoint LSPs is currently still in its draft version.

IETF Documents regarding CR-LDP:

CR-LDP (RFC 3212, Proposed Standard)
Applicability Statement for CR-LDP (RFC 3213, informational)
LSP Modification using CR-LDP (RFC 3214, Proposed Standard)
MPLS Working Group Decision on MPLS signalling protocols (RFC 3468, informational)

IETF Documents regarding RSVP:
RSVP (RFC 2205, Proposed Standard)
RSVP-TE (RFC 3209, Proposed Standard)
GMPLS Signalling RSVP-TE Extensions (RFC 3473, Proposed Standard)
Procedures for modifying RSVP (RFC 3936, BCP 96)
RSVP-TE Extensions in support of End-to-end GMPLS Recovery (RFC 4872, Proposed Standard)
Extensions to RSVP-TE for P2MP TE LSPs (RFC 4875, Proposed Standard)
Inter-domain MPLS and GMPLS TE extensions for RSVP-TE (RFC 5151, Proposed Standard)
Signalling RSVP-TE P2MP LSPs in an inter-domain environment (Draft as of March 2009)

1.5. Path Computation

Constraint-based path computation is a fundamental building block for traffic engineering systems such as Multiprotocol Label Switching (MPLS) and Generalized Multiprotocol Label Switching (GMPLS) networks. It is used to determine the path through the network that traffic should follow, and provides the route for each Label Switched Path (LSP) that is set up.

Thus, a PCE is an entity capable of computing complex paths for a single or set of services. A PCE might be a network node, network management station, or dedicated computational platform which is aware of the network resources and has the ability to consider multiple constraints for sophisticated path computation.

Nowadays, the model of the Internet is to distribute network functionality (e.g., routing) within the network. PCE is not intended to contradict this model and can be used to match the model exactly, for example, when the PCE functionality coexists with each Label Switching Router (LSR) in the network. PCE is also able to augment functionality in the network where the Internet model cannot supply adequate solutions, for example, where traffic engineering information is not exchanged between network domains.

Path computation in large, multi-domain, multi-region, or multi-layer networks is complex and may require special computational components and cooperation between the different network domains. These standards try to solve this kind of problems and it is important to see in what way they can contribute to the FEDERICA requirements.

Path Computation Element-based architecture (RFC 4655) specifies the architecture for a Path Computation Element (PCE)-based model to address the problem space. It describes a set of building blocks for the PCE architecture from which solutions may be constructed. For example, it discusses PCE-based implementations including composite, external, and multiple PCE path computation. Furthermore, it describes architectural considerations including centralized computation, distributed computation, synchronization, PCE discovery and load balancing, detection of PCE liveness, communication between Path Computation Clients (PCCs) and the PCE (PCC-PCE communication) and PCE-PCE communication, Traffic Engineering Database (TED) synchronization, stateful and stateless PCEs, monitoring, policy and confidentiality, and evaluation metrics.

The communication protocol between PCCs and PCEs (both elements described in the RFC 4655), and also between PCEs, are explained in the RFC 4657 (PCE Protocol Generic Requirements) Subsequent documents of this one will describe application-specific requirements for the PCE communication protocol.

RFC 4674 presents a set of requirements for a Path Computation Element (PCE) discovery mechanism that would allow a Path Computation Client (PCC) to discover dynamically and automatically a set of PCEs along with certain information relevant for PCE selection. It is intended that solutions that specify procedures and protocols or extensions to existing protocols for such PCE discovery satisfy these requirements.

PCE Communication Protocol (PCECP, RFC 5440) is an important issue inside Path Computation for communication between a Path Computation Client (PCC) and a PCE, or between two PCEs. Specific requirements for PCRCP are standardized: inter-area MPLS and GMPLS TE (RFC 4927) and inter-AS (RFC 5376).

When the PCE is a Label Switching Router (LSR) participating in the Interior Gateway Protocol (IGP), or even a server participating passively in the IGP, a simple and efficient way to announce PCEs consists of using IGP flooding. For that purpose, it defines extensions to the Open Shortest Path First (OSPF) routing protocol (explained above inside the routing section) for the advertisement of PCE Discovery information within an OSPF area or within the entire OSPF routing domain (RFC 5088). Also defines extensions to the IS-IS Protocol Extensions for PCE Discovery (RFC 5089).

Besides, the concept of policy was introduced in the context of path computation (RFC 5394). The document provides additional details on policy within the PCE architecture and also provides context for the support of PCE Policy.

Finally, RFC 5152 specifies a per-domain path computation technique for establishing inter-domain Traffic Engineering (TE) Multiprotocol Label Switching (MPLS) and Generalized MPLS (GMPLS) Label Switched Paths (LSPs). In this document, a domain refers to a collection of network elements within a common sphere of address

management or path computational responsibility such as Interior Gateway Protocol (IGP) areas and Autonomous Systems.

IETF Documents regarding path computation:
A Path Computation Element-based architecture (RFC 4655, Informational)
PCE Protocol Generic Requirements (RFC 4657, Informational)
OSPF Protocol Extensions for PCE Discovery (RFC 5088, Proposed Standard)
Requirements for PCE discovery (RFC 4674, Informational)
PCECP specific requirements for inter-area MPLS and GMPLS TE (RFC 4927, Informational)
IS-IS Protocol Extensions for PCE Discovery (RFC 5089, Proposed Standard)
Inter-AS requirements for the PCECP (RFC 5376, Informational)
Policy-enabled Path Computation Framework (RFC 5394, Informational)
PCE Communication Protocol (PCECP) (RFC 5440, Proposed Standard)
Per-domain path computation for establishing inter-domain TE LSP setup (RFC 5152, Proposed Standard)

Annex 2. Tools and Frameworks for further research

The deliverable JRA1.1 provides conclusions on which tools are appropriate for further investigation. These are BLUEnet Tool, DRAGON, IaaS Framework, and PL-VINI. This chapter will describe for each of the related topics, its relation to the standards mentioned in chapter 2.

2.1 BLUEnet Tool

Since topology discovery has not been standardized in the Internet community, BLUEnet tool can use any mechanism possible to include this feature in the tool. It uses two Perl scripts that run periodically every hour. These scripts discover routers (including adjacent links and connected switches), switches (including adjacencies, STP information, VLAN information, etc.). The script then logs on to the Topology Database and stores the obtained information in this database.

For routing, BLUEnet tool uses the link state protocol IS-IS. It is unknown in what way extensions of IS-IS are used within BLUEnet tool. This should be researched further.

For signalling, BLUEnet tool uses the Label Distribution Protocol. This means that it does not traffic engineering with signalling. Preferably, the tool should use at least constraint-based routing LDP (CR-LDP) or RSVP-TE. Since CR-LDP has been discarded by the IETF for further research, the signalling protocol used should be RSVP-TE.

The BLUEnet tool does not use the PCE architecture for path computation. Provisioning of Layer 2 links is done automatically, where the user only has to set two end points and the type of circuit (port mode or VLAN mode) required.

2.2 DRAGON

Topology discovery in DRAGON is done by listening to the OSPF-TE protocol. How this is done specifically should be investigated further. One approach could be the use of LSA reflectors and LSA aggregators, where adjacencies with routers are created to obtain a view of the topology. Inter-domain topology exchange can be based on the actual topology as obtained by listening to the OSPF-TE protocol, or based on an n abstracted view of the domain topology. This abstraction can be generated by a configuration file or by automatic synthesis of the link state database. This way, a simplified view of the topology can be seen at other domains.

In DRAGON, Network Aware Resource Brokers (NARBs) are implemented as agents that represent a single domain. It exchanges information with other NARBs, which present other domains. This enables end-to-end LSP routing. For routing, DRAGON uses a modified version of OSPF-TE. Each LSR must support at least an intra-domain version of GMPLS-OSPF. The NARB acts as a protocol listener in intra-domain routing, and is responsible for inter-domain routing.

The NARB also includes advanced algorithms which allow path computation with multiple constraints, such as the standard GMPLS-TE parameters, AAA constraints, scheduling, or vendor specific constraints. To implement TE, AAA, and scheduling constraints into path computation, DRAGON uses a 3D Resource Computation Element (3D RCE), which is located inside the NARB. The output of the path computation is an Explicit Route Object. Thus, for path computation DRAGON does not implement the PCE architecture as proposed by the IETF.

Each LSR must run an intra-domain GMPLS signalling protocol. This could be for example RSVP-TE or GMPLS UNI. In what way the extensions could be implemented must be further investigated.

2.3 IaaS Framework based Tools: Manticore, Argia.

Tools based on IaaS Framework are not able to discover automatically network topology. The physical administrator must introduce it manually using the management plane (through the graphical interface).

Regarding routing, Manticore, based on IaaS Framework, is capable to configure OSPF (link state protocol) and RIP for internal routing and BGP (inter-domain routing protocol). In which way Manticore is able to handle OSPF extensions must be researched further.

IaaS Framework has a Web Service based architecture. WS keeps connection between the different distributed components.

For path computation, Argia uses two algorithms for point-to-point paths. These are the Dijkstra shortest path algorithm, or an algorithm that computes all possible routes and gives the user the opportunity to choice the preferred path. Point-to-multipoint paths can also be found using two algorithms: a Dijkstra shortest path equivalent algorithm, or an algorithm that computes all possible routes and lets the user choose.

2.4 PL-VINI

In terms of network topology discovery this tool is not able to sense any underlying network changes. A solution must be found to add this functionality to the tool.

Concerning the routing aspects, each PL-VINI node can implement a XORP distribution. XORP implements a number of routing protocols, such as BGP, OSPF, RIP, IGMP, and MLD. There are some issues running different routing protocols simultaneously on the same physical interface. This possibility should be investigated further when defining the scope.

The signalling functionalities of PL-VINI are not clear, as they are not provided in any documentation available. This should be researched further by contacting the PlanetLab consortium.

PL-VINI does offer three methods for path finding. The first method is based on the locally available BGP table, which corresponds with the number of AS hops from the local node to the remote node. Counting AS hops is weakly correlated to actual latency, but might also be appropriate for applications that want to minimize the number of peering points that are traversed. The second method is that the local node runs traceroute to the target node and reports the number of router hops. As the router hop count is stronger correlated to the latency than the AS hop count, this method is more valuable for the computation. In the third method the local node can ping the target node and return the corresponding round-trip time.